On the Characterization of Library Cells

by

Jos Budi Sulistyo

Thesis Submitted to the Faculty
of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Dr. Dong S. Ha, Chairman
Dr. James R. Armstrong
Dr. F. Gail Gray

August 2000
Blacksburg, Virginia

Keywords: VLSI, Standard Cell, Characterization, Timing Model, Power Estimation, Synopsys, Cadence

On the Characterization of Library Cells

Jos Budi Sulistyo

Dr. Dong S. Ha, Chairman

Bradley Department of Electrical and Computer Engineering

(Abstract)

In this work, a simplified method for performing characterization of a standard cell is presented. The method presented here is based on Synopsys models of cell delay and power dissipation, in particular the linear delay model. This model is chosen as it allows rapid characterization with a modest number of simulations, while still achieving acceptable accuracy. Additionally, a guideline for developing standard cell libraries for use with Synopsys synthesis and simulation tools and Cadence Placement-and-Routing tools is presented. A cell layout library, built in accordance with the presented guidelines, was laid out, and a test chip, namely a dual 4-bit counter, was built using the library to demonstrate the suitability of the method.

Acknowledgements

Special thanks are due to my commitee chairman Dr. Dong S. Ha, through whose patience, understanding, and invaluable advice, this work has been accomplished. I would also like to express my gratitude to Dr. James R. Armstrong and Dr. F. Gail Gray for serving as my committee members.

I would also express my special thanks to the members and former members of the VTVT (Virginia Tech VLSI for Telecommunications) Group at the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, namely – in no particular order – Dr. Han Bin Kim, Suk Won Kim, Carrie Aust, Meenatchi Jagasivamani, Jia Fei, Nate August, Steve Richmond, and Andrew Gouldey, for all their help and suggestions along the course of this work.

Last but not least, my parents deserve a special kind of thanks. While they were not directly involved in this work, they nonetheless supported me with their encouragement, understanding and patience, particularly when I faced some particularly difficult problems. Without their support, it would not have been possible for me to accomplish this work. I could only wish I know how to thank them.

Contents

List of Figures

Chapter I

Introduction

As the complexity of circuit designs grows, it is becoming increasingly impractical to design logic circuits by hand. Therefore, the use of automatic synthesis tools has become mandatory.

In general, synthesis tool-based designs are performed using the following steps:

1. Description of circuit behavior in some high-level language, such as VHDL and Verilog
2. Compilation of behavioral description into a logical netlist using logic synthesis tools
3. Translation of the logical netlist into a geometric netlist, followed by placement and routing, with Placement-and-Routing (PNR) tools

The second step presumes that the design environment already contains some descriptions of some structural logic primitives (e.g. primitives for NAND gates, latches, flip-flops, etc), as those primitives will comprise the netlist produced by the synthesis tool. Similarly, the last step presumes that the translation of a netlist to geometric shapes is already defined for the design environment, i.e. the logic primitives referred to by the netlist is already present in some physical library. Hence, for the design environment, a library which contains both physical (i.e. layout) primitives and logic primitives which correspond to those structural primitives must already be present.

Therefore, with this design method, it is mandatory that a standard cell library be present. Further, the standard cell library should, at the minimum, consist of:

1. layout
2. other geometric descriptions as needed by the PNR tools, if the full layout is deemed too complicated for this purpose
3. list of logic primitives which correspond to those cells, including pinout
4. logic description libraries, both for synthesis and simulation purposes, which features simplified timing and power dissipation modeling capabilities

The last point deserves some clarification. While more accurate information (timing and power dissipation) could be obtained through the use of a commercially available circuit simulation program such as SPICE, the runtime tends to be prohibitively lengthy for large circuits. Further, at this design stage, it is often unnecessary to obtain, for instance, a power dissipation estimate which is accurate to within 5%. Hence, the use of simplified models, with their reduced accuracy but improved simulation speed, is the norm.

In this work, a set of methods for constructing such a library is presented. Chapter II will present a brief overview of the remaining parts; it will briefly describe the requirements imposed on standard cell libraries, as well as design process using such libraries and the development process to create the libraries. The chapter will also review several previous works on the characterization process. Chapter III will discuss some details regarding the development of the layout library. Chapters IV and V will discuss the characterization process to determine the cells' timing and power dissipation parameters in more details, while Chapter VI will present an example circuit – a 4-bit counter – constructed with the core cells developed in accordance to the steps discussed in previous chapters. Chapter VII summarizes the thesis.

Chapter II

Background

The use of synthesis and PNR tools requires the provision of a cell library. Therefore, the first step of the design is to develop such a library, or to acquire one. In this chapter, we provide background for development of a cell library.

2. 1. About Standard Cell Libraries

In theory any logic system could be built with AND and OR gates and inverters. However, to use synthesis and PNR (Placement and Routing) tools, there are some properties and features that a layout library must posses. The two most common of them are as follows.

The first requirement is the functional completeness. This depends on the synthesis tool in use, and is different from the functional completeness as ordinarily defined. Most synthesis tools require the presence of tristate elements. Also, they cannot ordinarily create sequential elements from combinational elements, or create a flip-flop from latches, and vice-versa. For example, Synopsys's Design Analyzer synthesis tool requires the library to contain, at the minimum, six different types of cells, namely:

- one type of tristate cell;
- either NOR and NAND gates, or AND and OR gates;
- inverter;
- D flip-flop with asynchronous set and reset;
- D latch with asynchronous set and reset

The second requirement pertains to the shapes and sizes of cells. The shapes and sizes of standard cells must be made very regular. This also applies to the geometries inside the cells, particularly those on metal layers. To facilitate routing, most PNR tools impose some requirement on the locations and the shapes of metal geometries inside the cell. Those

requirements are intended to ensure that PNR tools would be able to lay down routing tracks without being obstructed by metal geometries which form parts of the cell. In particular, since many routing tools are grid-based, i.e. they can lay down tracks only following some routing grids as defined by cell library designer, the metal tracks inside the cells need to be placed on those grids to facilitate the task of connecting those tracks to the outside world.

2. 2. Typical Standard Cell Based Design Flow

Typically, design of circuits based on standard cell libraries, starting from functional descriptions to completion of physical layout (prior to fabrication) follows the flow diagram shown in Figure 2.1.

From the flow diagram, several observations and comments could be made. Two of those observations and comments are as follows:

1. The library must contain descriptions of the cells to facilitate synthesis – specifically, translation into a netlist of logic primitives.
2. Since simulation of synthesized circuit may have to be performed repeatedly (e.g. if the speed or power dissipation does not satisfy goals), the models in logic primitives have to be simplified to allow fast simulation.

The second point implies that the library must contain information about timing and power dissipation parameters of the cells. For Synopsys tools, these are provided by the library designer in the form of library (.lib) file. This file is then compiled into both synthesis library (for synthesis step) and simulation library (for simulating synthesized circuit, or also to be invoked directly in design if structural design is attempted).

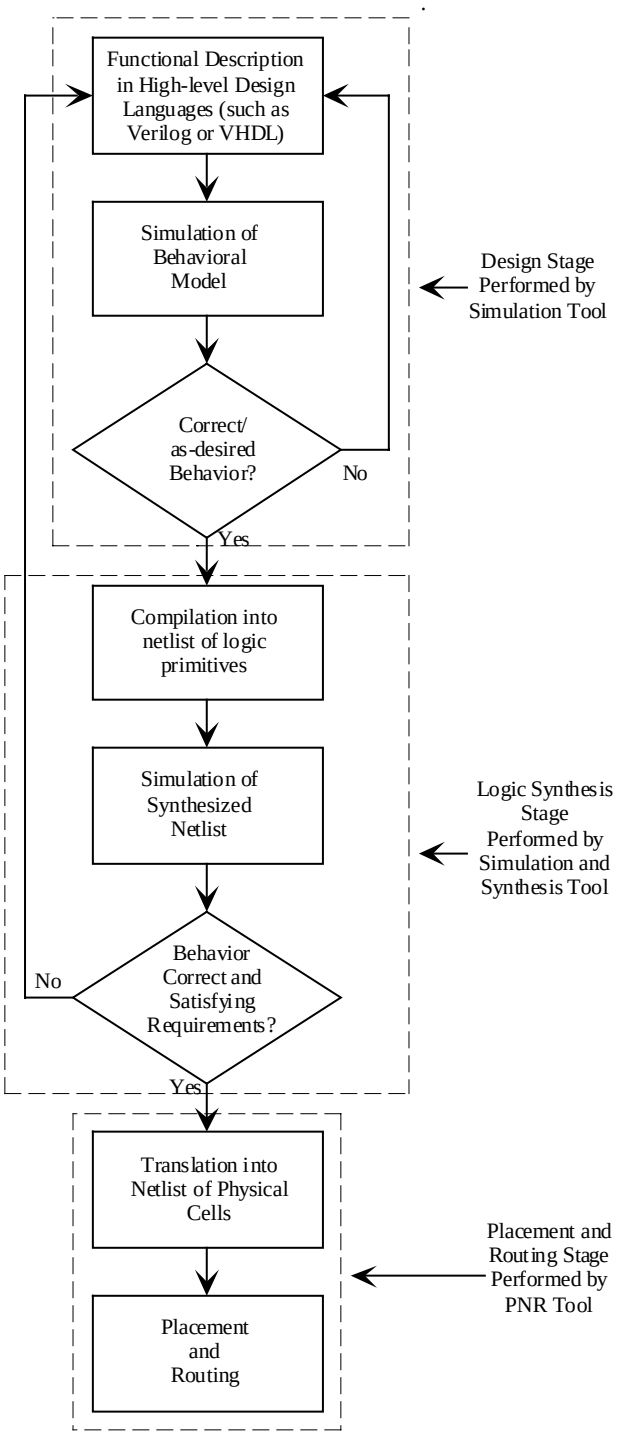


Figure 2.1. Flow Diagram of Standard Cell Based Design

Additionally, since the tasks of the PNR tools are simply properly placing the cells and laying down interconnects, it is unnecessary for the PNR step to use complete layout. A

simplified representation, which include only routing layers and other layers which poses significant electrical contact with the routing layers (e.g. includes only metals, contacts, and vias) would be adequate, as long as it corresponds to an actual layout. Also, such a simplified model should be faster to process. For Cadence Silicon Ensemble PNR tool, this is provided in form of LEF (Library Exchange Format) file, which contains a partial description of all cells in the library, some design rules pertinent to placement and routing process (such as metal and via spacing), and the routing rules defined by the designer of the library (such as the desired pitch between two adjacent metal tracks).

2. 3. Standard Cell Library Development Process Flow

Typically, the development of a standard cell library, starting from layout to porting to simulation, synthesis, and PNR libraries, is as shown in Figure 2.2.

Basically, the development process consists of three basic steps: netlist- (or schematic) level design, layout, and porting to synthesis/simulation libraries and to PNR library. Returning to the standard cell-based design process shown in Figure 2.1, the synthesis is to be used in the logic synthesis stage; the simulation library is to be used during synthesis and sometimes also design stage; and the PNR library is to be used in PNR stage.

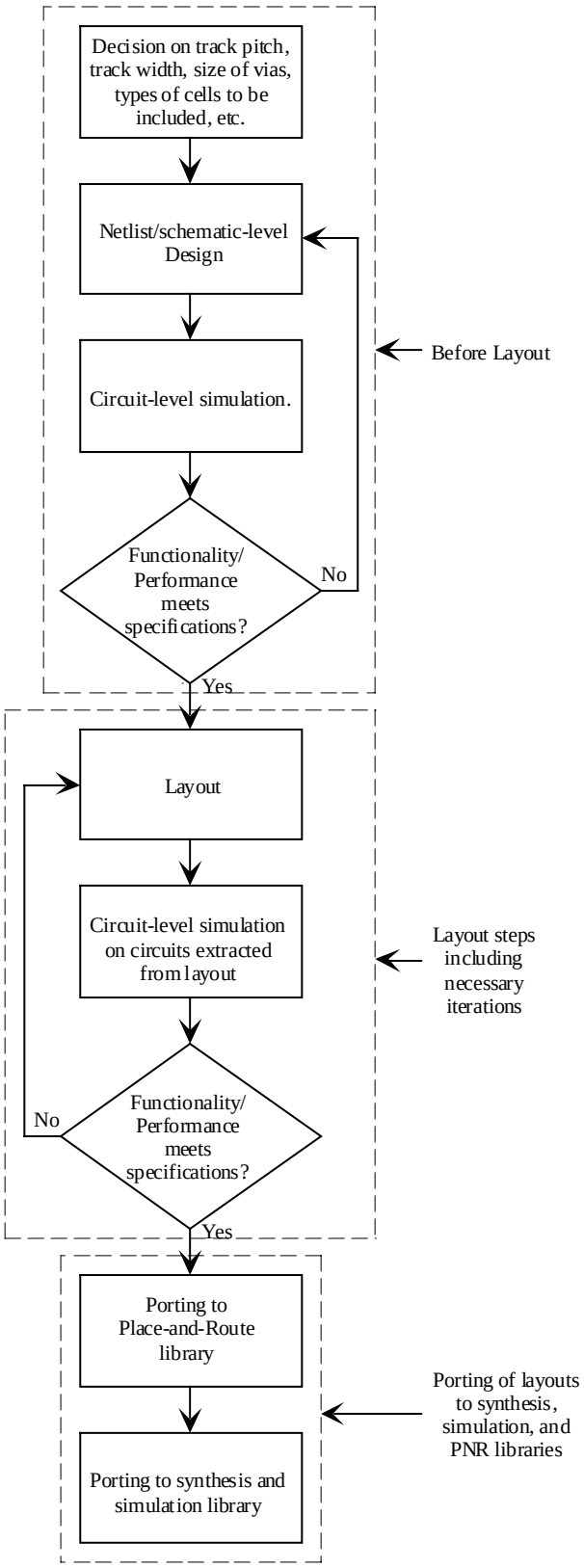


Figure 2.2. Flow Diagram of Standard Cell Library Development Process

2. 4. Review of Previous Work

As most of the rest of this work will be concerned with devising a relatively simple yet relatively accurate characterization method, this section is primarily dedicated to reviewing previous works on cell characterization. In particular, since the linear timing model used by Synopsys tools is largely based on Penfield-Rubinstein-slope model, most of the works on timing characterization discussed here will be the ones which focuses on the particular model.

In [8], Patel proposed a method to characterize cell delay and capacitance parameters, and described a system implementing the method. Deserving a special attention is his proposed technique to determine actual switching voltages of each cell, as well as proposed delay definition as switching-to-switching voltage instead of, for instance, the more commonly used 50%-to-50% delay. While the proposed definition may result in a more accurate delay estimation for one kind of cell, it would likely lead into inconsistent definitions for cases where a cell drives another cell of different type (and hence, different switching voltage). While the use of 50%-to-50% delay is largely arbitrary, one virtue of this definition is that it is likely to result in a more consistent definition.

Further, in [3], Cirit proposed a similar method, which slightly differs in that it assumes the cell being characterized as a black box (i.e. making no assumption about its internal structure). While several details of the proposed methods are no longer applicable as the current versions of SPICE offer various new capabilities, the black-box approach proposed is largely adopted in our present work due to its simplicity.

The cell delay and power dissipation models used in this work is the one described in Chapters I and II of [9]. Specifically, for timing characterization, the linear timing model is used due to its simplicity and due to the relatively small numbers of simulations needed to characterize cells with acceptable accuracy. Also, in [5], Eshraghian and Weste describes several delay model, one of which, namely the Penfield-Rubenstein-slope model, appears to be essentially identical with the linear delay model used here.

Jou et al. in [6] and [7] proposed techniques to simplify characterization tables for complicated cells. The proposed techniques are particularly useful for cases where the internal structures of the cells are known.

Most of the previous works assumed some internal structure for the cells. In our work, cells are basically viewed as black box entities. Combined with relatively simpler types of cells encountered in a typical standard cell library, a more exhaustive power characterization method is possible, which would also allow a higher accuracy.

Chapter III

Development of Layout and Abstract Library

For a cell layout library to be properly usable in standard cell-based design, several requirements have to be satisfied. In general, those requirements depend on the particular synthesis tool and PNR tool used. Therefore, in this chapter, we will focus on the development process of cell library for use with Synopsys Design Analyzer synthesis tool and Silicon Ensemble / Envisia Silicon Ensemble family of PNR tools.

3.1. Requirements on the library

As previously suggested in Figure 2.2, the development process of a standard cell library for use with Silicon Ensemble (SE) routing tool follow the following steps:

1. Layout of cells
2. Creation of Synopsys synthesis and simulation libraries
3. Generation of LEF (Library Exchange Format) description of the cells

The LEF file is an ASCII file containing a partial geometric description of the cells in the library. This file is used by the SE routing tool during the placement and routing process. The description in the LEF file is simplified by including only metal layers and other layers which may obstruct routing, such as metal tracks inside the cells. It is unnecessary to precisely model the shape of the n-well in a detailed fashion, as the metal tracks laid down by SE should not be electrically influenced by the well to a significant degree.

This chapter has two objectives. The first objective is to outline a layout method which will result in logic cells which could be successfully converted to LEF description and could be successfully used by SE, and the second is to give several details of the LEF file generation steps which need to be followed to guarantee that the resulting LEF files could be successfully used with SE.

3.2. Layout Technique

3.2.1. What Silicon Ensemble does

Silicon Ensemble is a placement-and-routing (PNR) tool which accepts the Verilog netlist of the design to be physically synthesized as input, and with information regarding the shape and obstructions posed by each cell in the library it will construct the completed circuit.

It is assumed here that the reader is familiar with layout process.

Standard cells for use with Silicon Ensemble are drawn like any other cells, except that to facilitate routing process, the locations of routing layer shapes inside the cell has to be regular.

First, several terms need to be defined. Most of the layers in a rectangular cell could be grouped into three mutually exclusive groups: *routing layers*, *cut layers*, and *masterslice layers*. A *routing layer* is a layer in which the routing software performs routing by laying down wires or other shapes in that layer. Usually the routing is limited to metal layers – or even to just some of the available metal layers. A library must have at least one routing layer, and ideally should be many. A *masterslice layer* is the layer in which no routing is performed and which lies somewhere under the lowest routing layer. The routing tool will not lay any tracks in a masterslice layer. Typically these layers include n- / p-well, active regions, and poly, although in some cases it may be expedient to use poly in routing instead. It is allowable to use masterslice layers, such as poly, in intracell routing. However, it is not allowable to perform intercell routing in a masterslice layer. A *cut layer* is a layer which is used to connect two different routing layers or to connect a routing layer to a masterslice layer; via and poly-to-metal contact layers are examples of cut layers.

The SE place-and-route tool performs the following tasks. Upon receiving the Verilog netlist of the design, it lays down rows of cells, as shown in Figures 3.1 and 3.2.

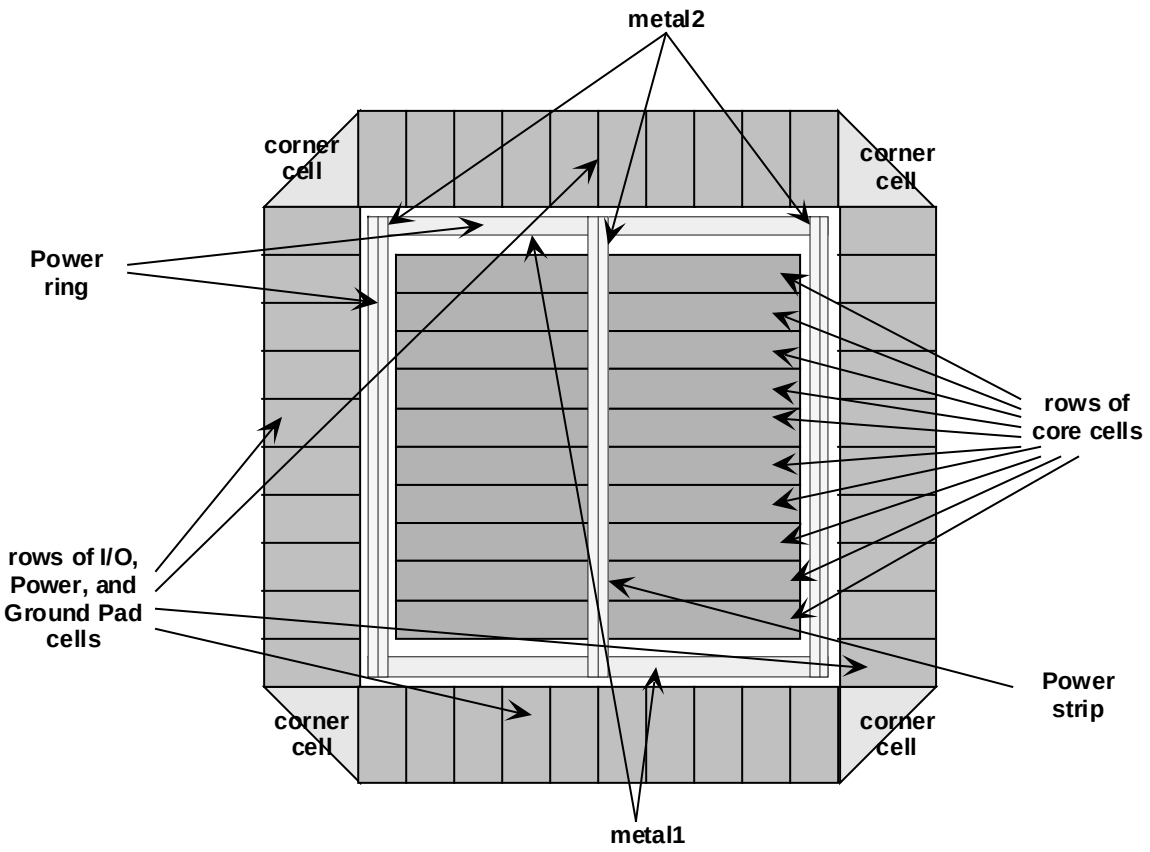
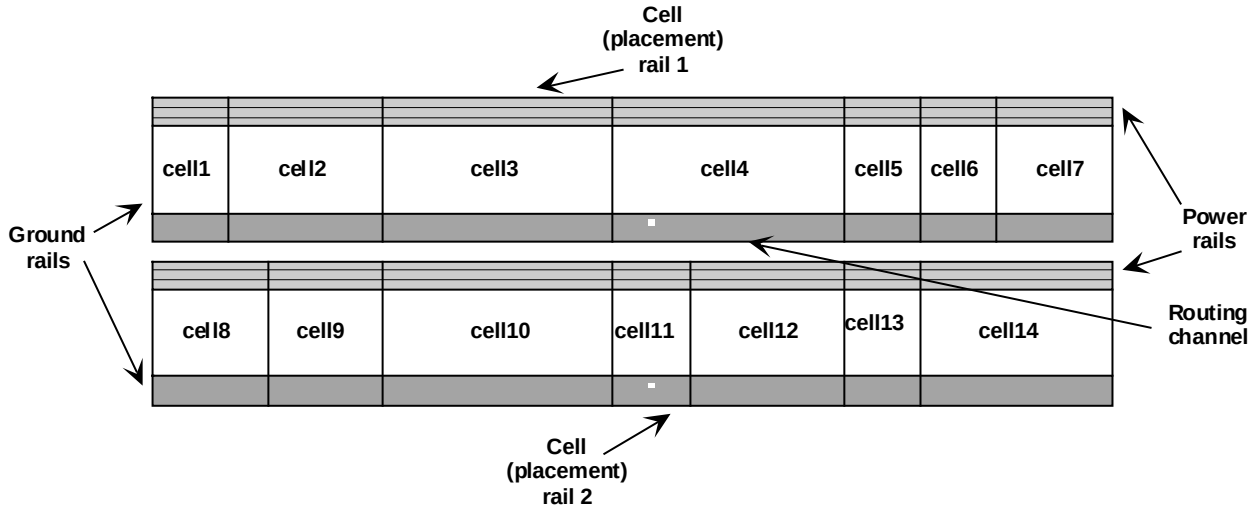


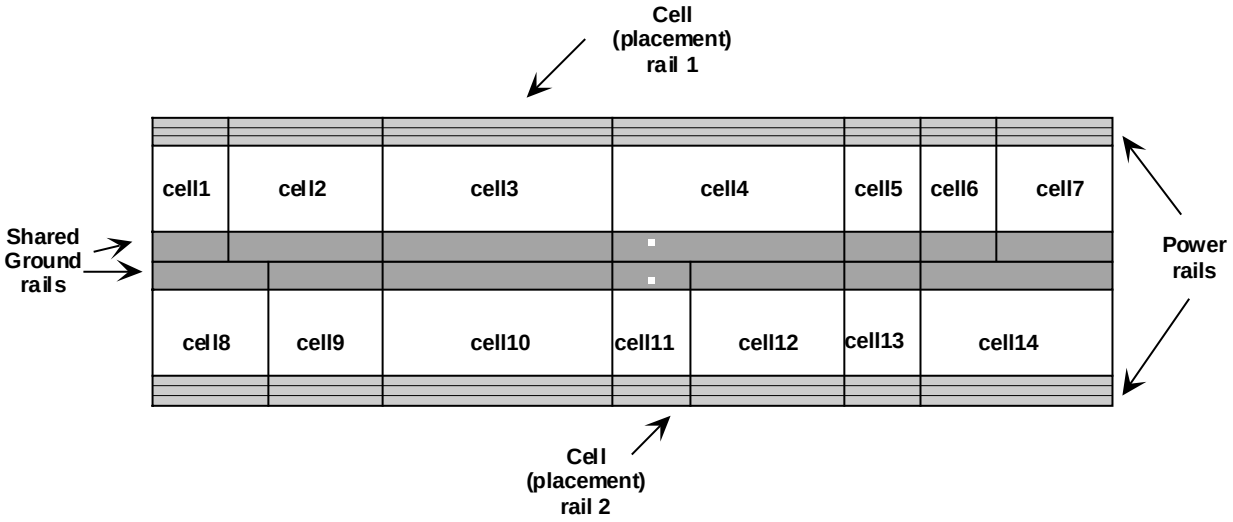
Figure 3.1. The general layout of a complete SE-routed chip

As Figure 3.1 shows, a completed chip would contain three different classes of cells: core (logic) cells, corner cells, and pad cells (I/O, Power, Ground). In this chapter, we are concerned only with the design of core cells.

Further, after the cells are laid down, SE connects the cells according to the netlist specified in the Verilog file. The connection is made over the top of the cell using upper-level metals (e.g. usually metal3-metal5 in a five-metal technology), but all metal layers should be declared as available for routing by the library designers, and the routing tool instead of the library designers should decide which metal layers to use.



(a) ***without flipping every other row***



(b) ***with every other row flipped***

Figure 3. 2. Two adjacent rows of core cells. With every other row flipped, rows can be abutted, which reduces both chip area and power wires resistance.

3.2.2. Requirements on the Layout Style

Silicon Ensemble has both grid-based and gridless router. Unlike the grid-based router, the gridless router is capable of routing pins placed in an almost arbitrary fashion, as long as there is space to place via. However, since the grid-based router is faster and is more guaranteed to succeed, the standard cells in use should exhibit the following properties:

The sizes, shapes and locations of all geometries in layers pertinent to routing are regularized. If, for example, a metal1 signal track inside the cell is 1μm wide, all other metal1 tracks inside the cell must also be 1μm wide.

- The general shape of the cell is as follows:

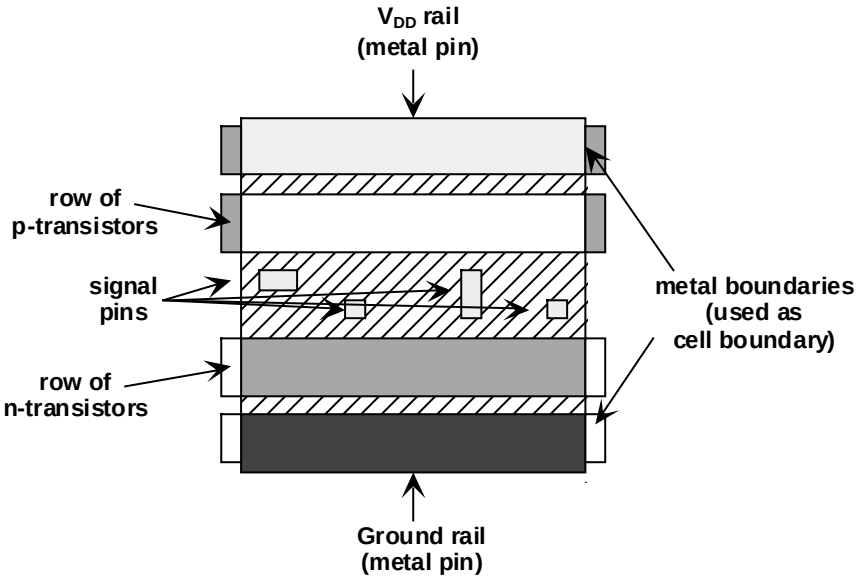


Figure 3.3. General shape of a standard core cell. Note that the term pins" refers to any shapes in the particular layer being used for routing.

- All metal tracks of the same layer (metal1, metal2, etc) for the same purpose (signal or power) must have the same width. If, for example, a metal1 track for signal connection is 0.5μm wide, then all other signal connections in metal1 in the library must also be 0.5μm wide. If a metal1 power pin is 2μm wide, all cells in the library must use 2μm wide metal1 power connections.
- All power / ground pins should have the same width and should run in same directions – all horizontal or all vertical.
- Power / ground pins should be in the form of rail at the top/bottom ends of cell.
- Attempts must be made to lay signal tracks of the same layer in the same direction.
- For any two adjacent signal track in the same metal layer running in the same direction, center-to-center pitch (defined below) must be either the same, or an integer multiply of a minimal pitch value (called *routing pitch*).

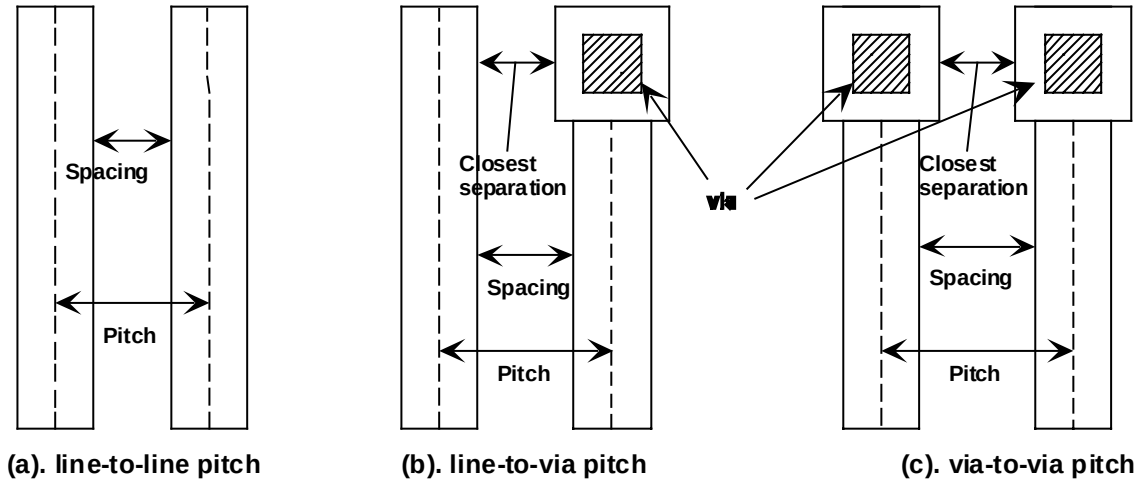


Figure 3.4. Definition of Routing Pitch

- The routing pitch should at least line-to-via pitch, as defined in (b), where the closest separation (line to metal extension of via) still satisfies design rule for metal-to-metal separation. Ideally, it should be at least via-to-via pitch (see picture (c)). This will allow the routing tool to drop via where necessary. Avoid using only line-to-line pitch as in (a), as the routing tool may fail since it is unable to drop a via when it is needed.
- The width of metal layers in user's unit should be divisible by two, to facilitate the routing tool to mirror the cell when it is needed.
- All available of metal layers should follow the rules, even if there is no intention to actually use it for routing – for example, in a ten-metal-layer process where it may not be critical that the first layer be usable. It should be possible to declare all metal layers as available for routing during the LEF file generation later on, and it will allow the routing tool decides which metal layers will actually be used, which will likely result in better routing than if the cell designer explicitly prohibits the use of the first metal layer.
- In a multi-metal process, if for any reason the routing pitch is not identical for all metal layers, then the ratio of pitch between any two metal layers should be kept simple, such as 2:1 or 3:2 (ideally should be 1:1 if possible). Complex ratios, such as 11:9, should be avoided.

– Regardless of the number of metal layers provided by the technology, the number of metal layers used for internal connections within the cells should be limited. If possible, limit the metal track use to metal1 only, so that all higher metal layer tracks are freely available for use by the routing tool.
– The distance, both in x- and y-direction, from the cell's metal corner to any of the centers of via or metal shapes must be integer multiplies of routing pitch of the metal layer with the smallest pitch (typically metal1). This will help the metal layers, and the pins defined in the metal layers, to have consistent offset values (defined as follows).

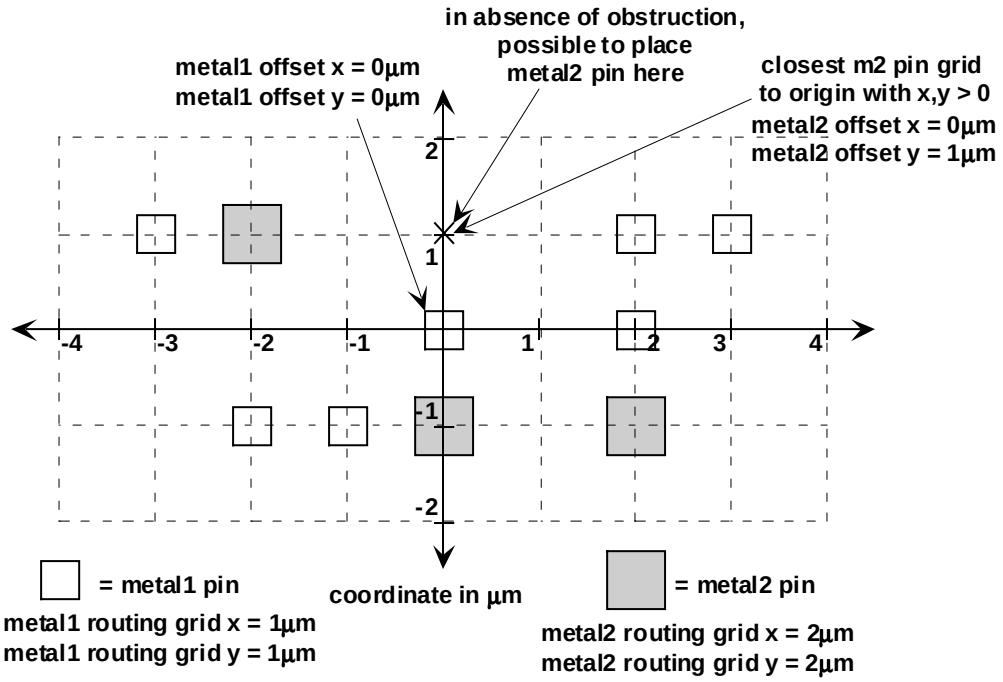


Figure 3.5. Definition of offset. It is assumed here that all pins are on-grid – they are all located in the grids specified in the technology LEF

Offset is the distance (x-offset and y-offset for x- and y-distance, respectively) from origin to, or the xy-coordinate of, the center of the possible pin location (as defined by the grids) which is closest to origin and has positive coordinate in both x- and y-directions (*see ilustration*).

A special note is regarding the ratio among routing pitch of different metal layers. It is tempting to think that since cell sizes are generally minimized if smallest metal widths and pitches are used, this will be true of the overall circuit size. However, this is not necessarily the

case. As a case in point, the following picture compares the difficulty of laying a via in a 3:2 pitch ratio (here metal2 and metal3 have pitch values of 1.5× pitch for metal1) as compared with the case where we have 1:1 pitch.

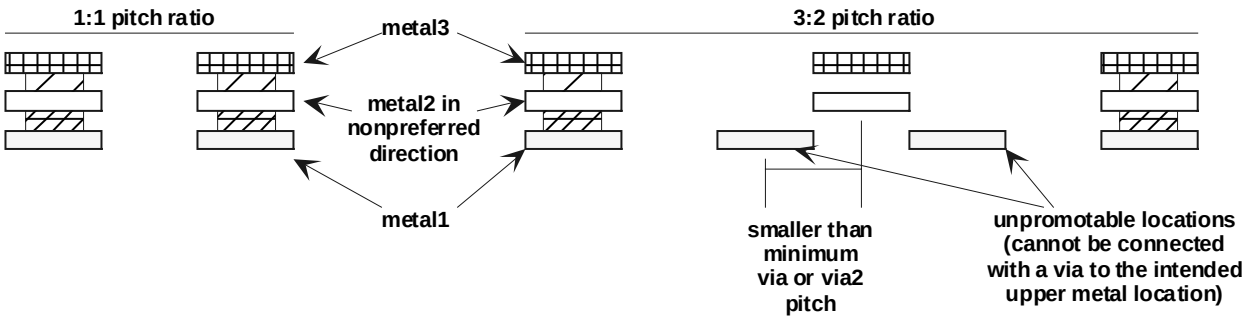


Figure 3.6. Example of placing two stacked vias for two different pitch ratios. The more complex the pitch ratio, the larger percentage of all available tracks cannot be connected with via.

As apparent from the example, the disadvantage of a 3:2 or 4:3 pitch ratio as opposed to 1:1 is as follows. Suppose the obstructions posed by the tracks inside the cell necessitates the formation of parallel metal1-metal2 tracks, and metal1-metal2 connections need to be made. For a 1:1 track ratio, it would be easy to use all available metal1 and metal2 tracks and simply drop the via to connect them. For a 3:2 ratio, however, two metal1 tracks, or one metal2 tracks, will have to be spent just to make one connection. In this case, minimizing the pitch alone will not increase the density of the routed circuit. Even worse if the pitch ratio is, for example, 4:3. In general, 1:1 is ideal, 2:1 is reasonable, and 3:2 is the worst still considered acceptable.

The rules previously discussed are necessitated by the way SE performs routing. The tool performs routing by laying down horizontal, vertical, and Manhattan-style tracks. For each metal layer, the direction could be horizontal or vertical, but one direction is always taken as the preferred direction, and the other one is automatically non-preferred. For example, if for metal1, horizontal direction is considered preferred, the tool automatically tries to create horizontal tracks first before resorting to creation of vertical tracks, although vertical tracks will eventually be used if deemed necessary.

3. 2. 3. Example of Cell

An example of standard cell so constructed is the following D latch:

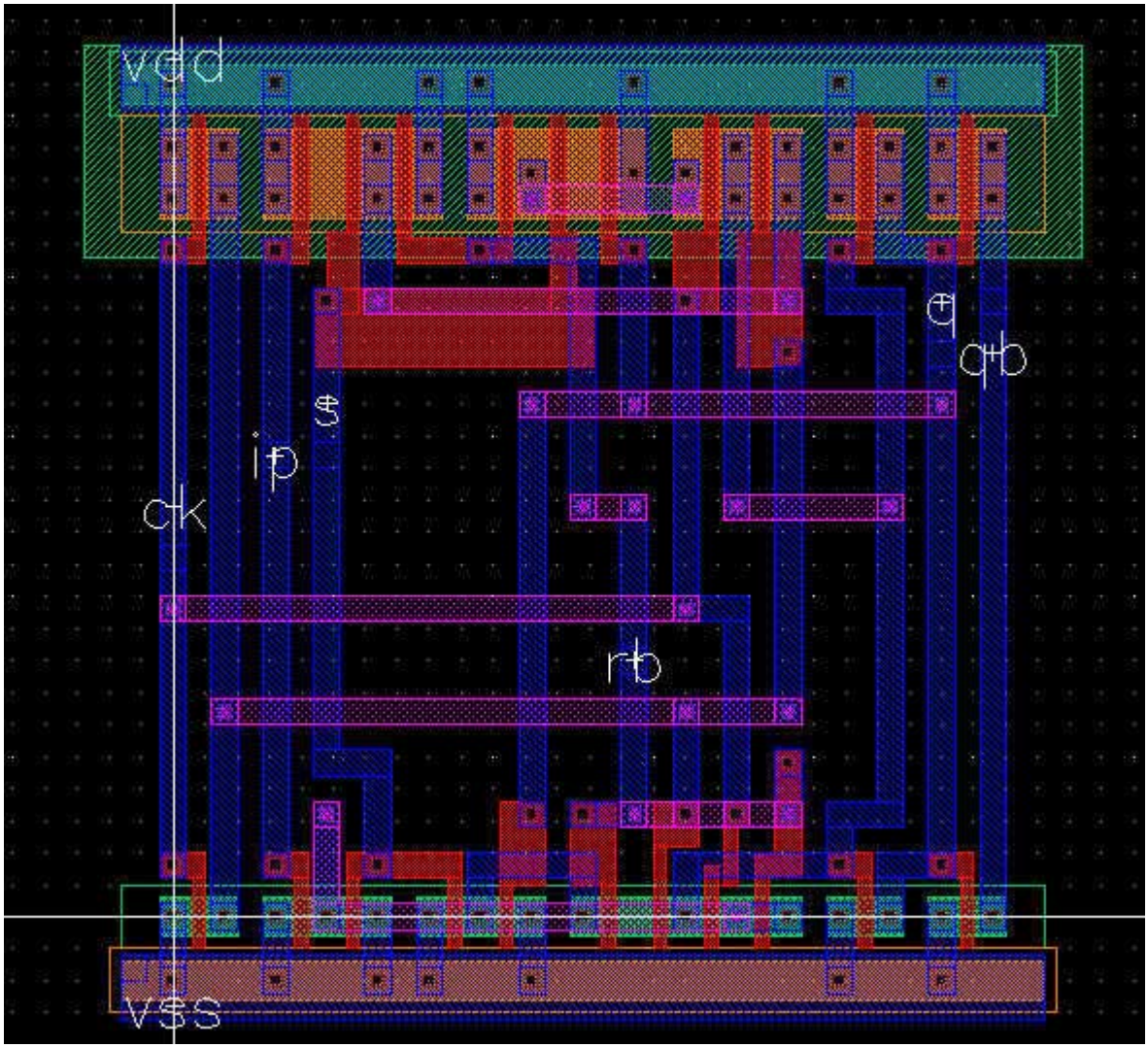


Figure 3.7. A high-active D latch with asynchronous set and reset.

Some parameters of the cell:

Cell Width = 28.8μm

Cell Height = 30.4μm (= 19× metal1 pitch)

Coordinate of metal1 lower left corner = (-1.6μm, -3.2μm) = (-1, -2) × metal1 pitch

metal1 width = 0.8μm metal3 pitch = 1.6μm (minimum = 0.6μm width, 1.2μm pitch)

metal2 width = 0.8μm metal3 pitch = 1.6μm

metal3 width = 0.8μm metal3 pitch = 1.6μm

metal4 width =1.2μm metal4 pitch = 3.2μm

preferred directions: metal1,3 = horizontal, metal2,4 = vertical

Note that although the pitch for metal1 could have been reduced (minimal spacing and minimal width are 0.6μm each), the chosen values are considered better in that they allow the same pitch is used for metal2. Had minimal width and spacing been used for metal1, a pitch of 1.2μm would have occurred, and a metal1 vs. metal2 pitch ratio of 4:3 would arise.

Note also that the boundaries of the cell – which determine how closely cells could be placed next to each other – are purposely based on metal tracks. In particular, the width of the cell is defined – during LEF file generation – as the same as the length of power tracks. The idea is that the power pins could simply be abutted to the power pins of adjacent cells, and the n- and p-wells those pins are connected to could also be abutted in like manner. Therefore, even though defining cell width as the same as power track length would cause parts of the wells to lie outside cell boundaries, it should not cause the routed cells to contain any design rules violations.

Additionally, the width of the cell is deliberately planned to be an integer multiply of metal1 width. This would guarantee the continuity of routing grids between any two adjacent cells.

Chapter IV

Timing and Capacitance Characterization

In this chapter, characterization of timing arcs and capacitance parameters of logic cells will be discussed. The term *timing arcs* refer to durations incurred between an input and its associated outputs, which is known as *delay*, as well as timing requirements pertaining to one or more inputs, or also known as *constraint*, such as setup time, which is a constraint pertaining to the duration between a clock signal and another input signal of a sequential cell. For purpose of characterization, both delays and constraints could be treated in the same manner. While capacitance is a distinct entity from delays, it is discussed here as well due to its influence on circuit delays. A delay model, which simplifies calculations of delays exhibited by a cell under a certain condition, will be discussed. The linear delay model will be used here due to its simplicity and common use, including in Synopsys synthesis tools. Further, a method to characterize capacitance and timing parameters will also be presented.

4. 1. Timing Model and Aim of Characterization

Although timing arcs actually include both constraints and delays, here we will focus more on characterizing delays for two reasons. First, delay is exhibited by all cells, while constraints are generally considered as belonging to sequential circuits only. It is uncommon that constraints related to combinational cells, such as minimal pulse width, need to be characterized. Second, while some timing constraints, such as setup times, are significant, delays tend to be the factors which plays a larger part in determining the highest speed attainable by a circuit. This is due to the fact that the sped of a circuit is generally determined by its critical path, which generally includes a rather large number of combinational circuit stages.

4. 1. 1. Basic CMOS Timing Model and Definition of Terms

In general, most of the simplified timing models used by high-level use the following mathematical model:

Delay of a cell = Intrinsic delay + transition delay + slope delay (4.1)

The meaning of the above terms will be explained as follows:

*The **intrinsic delay** of a cell is defined as the propagation delay of the cell without load, when it is driven by another identical loadless cell*. Both the driving cell and the driven cell (whose intrinsic delay is measured) should be loadless. The requirement necessitates that the driving cell derive the driven cell indirectly, as illustrated in Figure 4.1.

Note that Vin2 is a voltage source controlled by the output voltage Vout1. The intrinsic rise delay and intrinsic fall delays may be different; delays from each input pin to each output pin may be different from others. Hence, there are six different intrinsic delays, three intrinsic rise delays for the three inputs and three intrinsic fall delays, for the circuit in Figure 4.1.

***Transition delay** is defined as the additional delay* (in addition to intrinsic delays) *of a cell driving a capacitive load, but which is driven by another identical loadless cell*. The instantiation of the cells for the transition delay measurement is shown in Figure 4.2. This transition delay occurs because, as the result of a cell having to drive a capacitive load, its output slope becomes less steep (i.e. rise and fall times increase) compared with the loadless case.

***Slope delay** is an extra delay* (in addition to intrinsic and possibly transition delays) *of a loadless cell which is driven by an identical cell with transition delay*. The driving cell drives a capacitive load (and so it exhibits transition delay), and hence the output slope of the driving cell

is less steep than the one without a load capacitance. The less steep slope causes additional delay for the driven cell being evaluated.

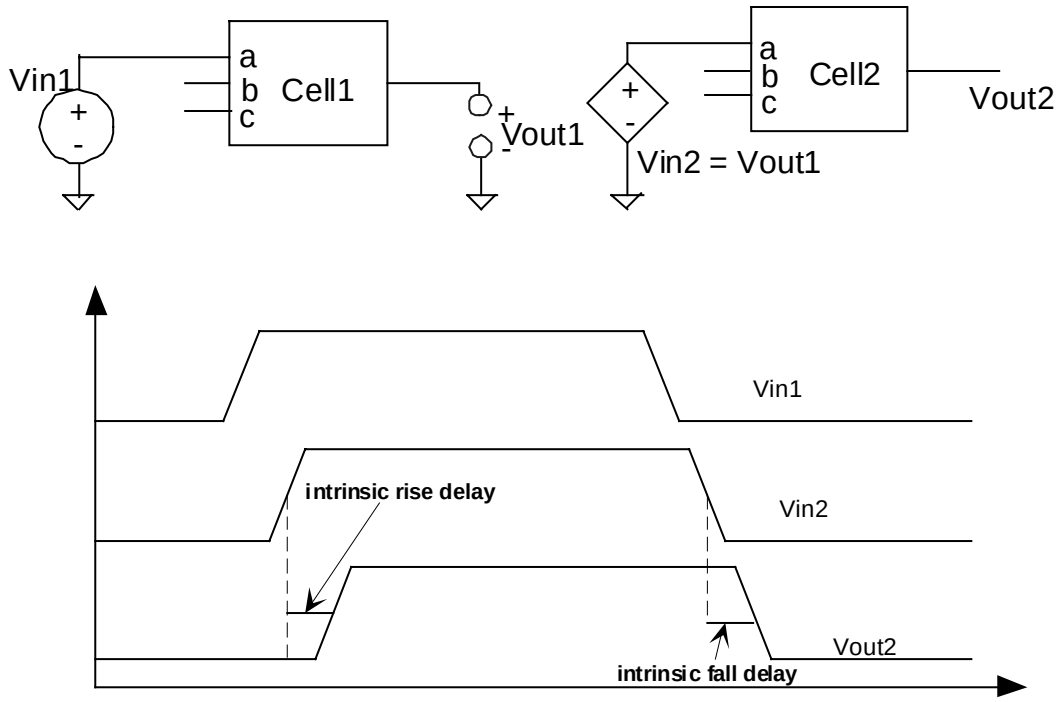


Figure 4.1. Definition of intrinsic delay. **Top:** Circuit definition, with Cell1, Cell2 being the same type. Vin1 is considered as having the same rise/fall time as an unloaded Cell1/2. **Bottom:** Waveform.

In reality, a cell will likely exhibit both transition and slope delay. The separation of slope and transition delay is an artificial construct, as it necessitates the use of ideal dependent voltage source, which is just a mathematical tool. However, in the method proposed later in this work, this approach will be followed as it provides a relatively easy means of characterizing various timing parameters of a cell.

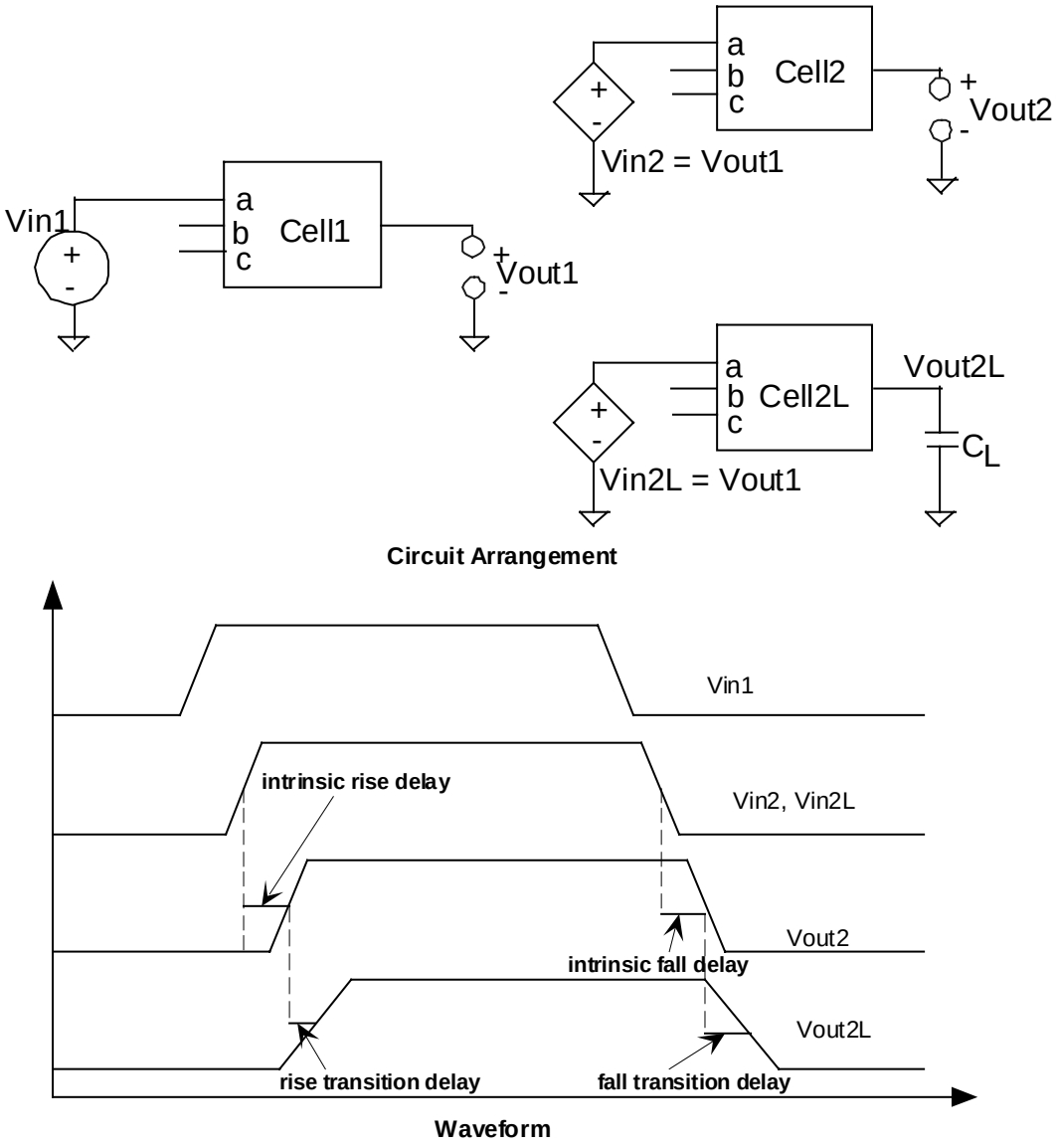


Figure 4.2. Definition of transition delay. **Top:** Circuit arrangement. **Bottom:** Waveforms. Note that Vout2L is delayed relative to Vout2, mostly due to the less steep slope due to the presence of capacitive load.

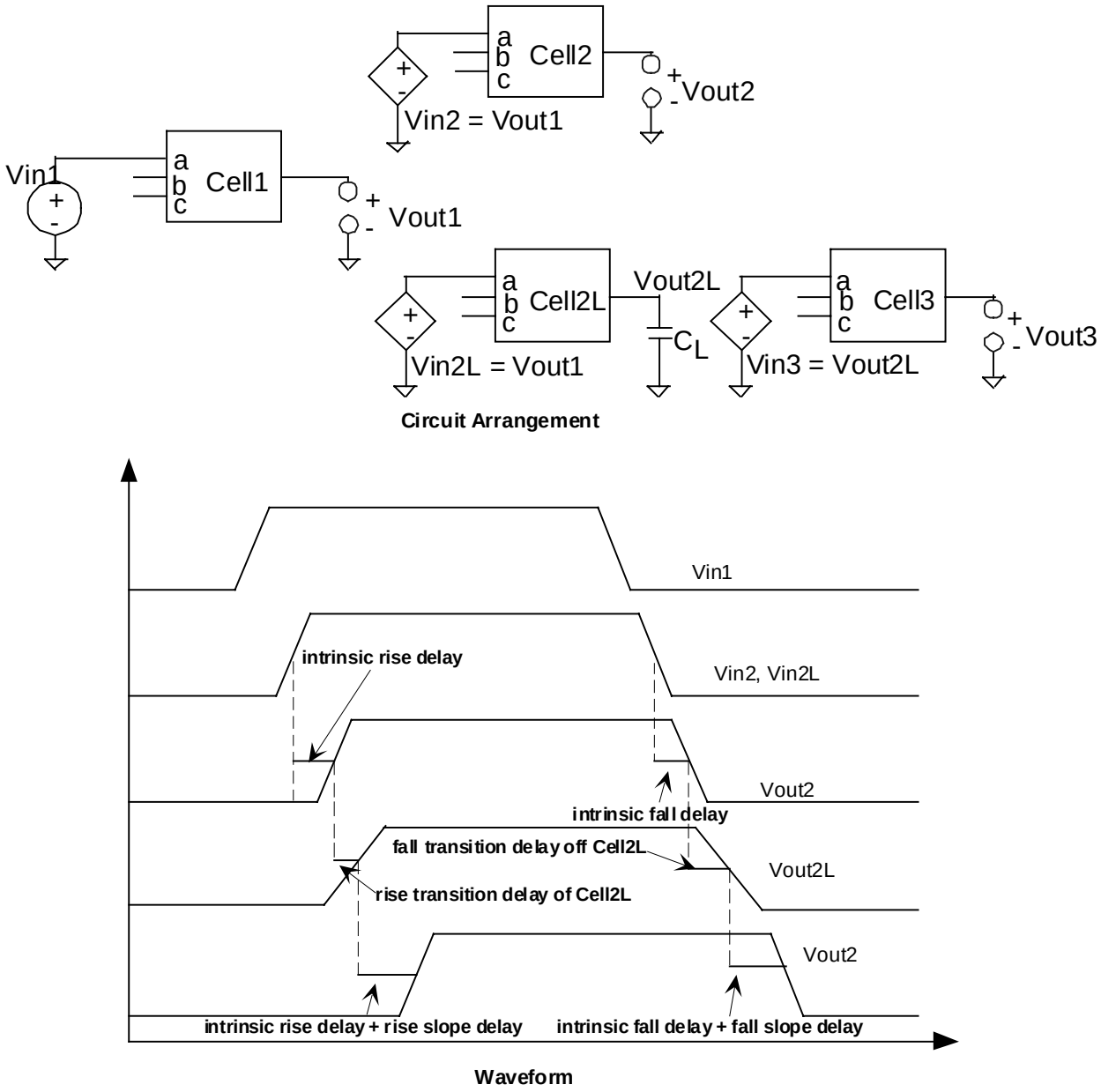


Figure 4.3. Definition of slope delay. **Top:** Circuit arrangement. **Bottom:** Waveforms. Note that the delay from Vout2L to Vout3 is longer than the delay from Vout1 (=Vin2) to Vout2. Since the slope of Vout2L is less steep than the slope of Vout1, Cell3 is slower to respond to Vout2L than Cell2 to Vout1.

4.1.2. Linear Delay Model

The mathematically simplest form of delay model which is built around Equation (4.1) is the one known as Linear Delay Model. In this model, transition delay is modeled to be linearly proportional to load capacitance, while slope delay is modeled as linearly proportional to the transition delay of the driving waveform:

transition delay = output resistance × load capacitance (4.2)

slope delay = slope sensitivity × transition delay of input waveform (4.3)

For example, for an inverter, in which a rising input triggers a falling output and vice versa,

rising transition delay = rising output resistance × load capacitance (4.4)

falling transition delay = falling output resistance × load capacitance (4.5)

rising slope delay = fall slope sensitivity × fall transition delay of input waveform (4.6)

falling slope delay = rise slope sensitivity × rise transition delay of input wavefor (4.7)

It must be understood here that the terms *output resistance* simply refers to some stipulated linearity factor. It does not really refer to *resistance* as normally defined. Normally, the resistance is defined as the derivative change in current with respect to change in voltage as a voltage stimulus is applied to a node, here to the output of a cell.

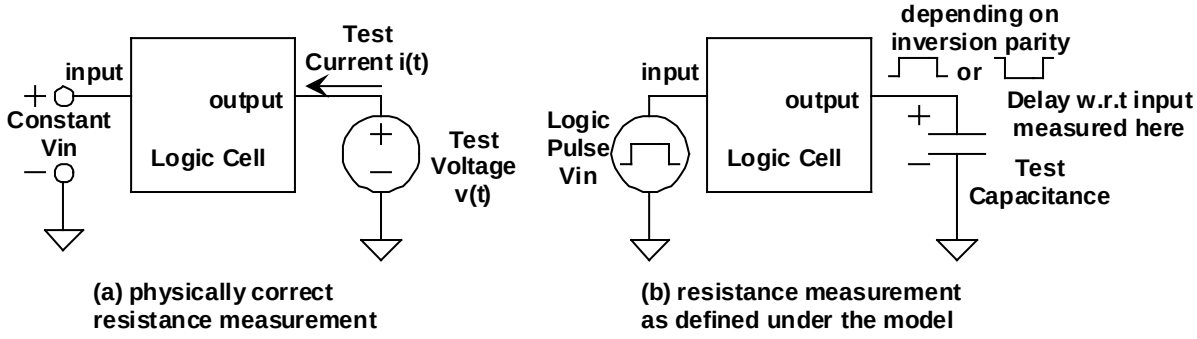


Figure 4.4. Output resistance measurement setup: difference between the physical definition and the definition of the model.

As shown in Figure 4.4.(a), the physical definition of output resistance is:

$$R_{out} = d(I_{test}) / d(V_{test})|_{Vin=constant} \quad (4.8)$$

Further, the test voltage must be slowly varying enough so that the output reactance does not figure into the equation. In contrast, for our model,

$$R_{out} = \Delta(delay)/\Delta(capacitance) \quad (4.9)$$

and we have different value of resistance for rising and falling output. In the linear model used by Synopsys tools, in fact, there is a separate pair of rising and falling resistance values for each combination of input and output pins. Even for the same output pin, the resistance may differ depending on which input pin triggers the output transition. Also, while the physical definition of resistance stipulates that there should be a unique value for resistance for each pin, the value of resistance (as well as intrinsic delay and slope sensitivities) may be defined in non-unique ways.

For instance, let us consider the case of a D flip-flop with asynchronous high-active set/asynchronous high-active reset as follows:

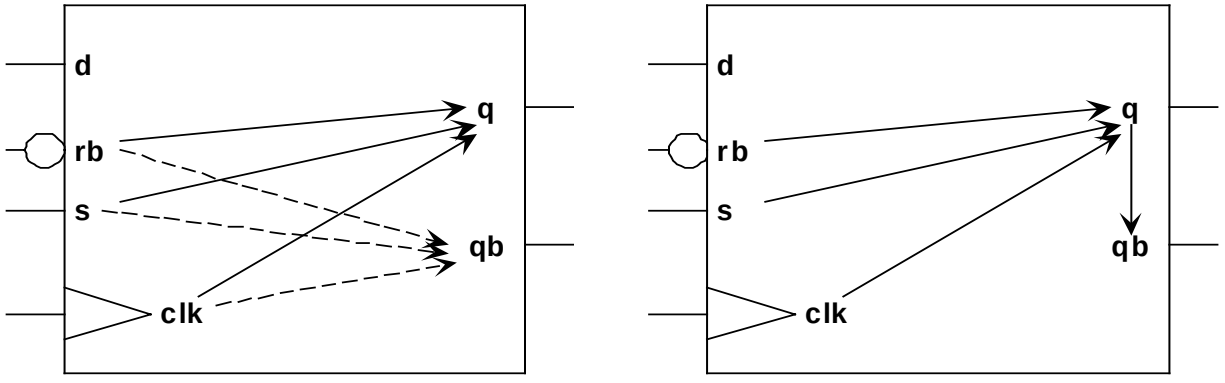


Figure 4.5 – Two different alternative arrangements of timing paths inside a D flip-flop with asynchronous set-reset. Arrangement (a) is more general but if the output qb is generated by inverting q internally, arrangement (b) is likely to result in a more accurate model.

Ignoring the possibility that the flip-flop will be set and reset at the same time (i.e. rb = 0 and s = 1 at the same time), we have the following paths:

For arrangement (a):

- clk rising to q rising
- clk rising to q falling
- clk rising to qb rising
- clk rising to qb falling
- rb falling to q falling
- rb falling to qb rising
- s rising to q rising
- s rising to qb falling

For arrangement (b):

- clk rising to q rising
- clk rising to q falling
- rb falling to q falling
- s rising to q rising
- q rising to qb falling
- q falling to qb rising

Therefore, for arrangement (a), we will have eight different intrinsic delays, eight different resistances, and eight different slope sensitivities, while for alternative (b), there will be six different intrinsic delays, six different resistances, and six different slope sensitivities. Note also that both models have their respective strengths and shortcomings. Alternative (a) is more general; it applies for the flip-flop regardless of the internal implementation; alternative (b) is useful for modeling only is qb is actually generated by inverting q. However, if this is indeed the case, alternative (b) will likely result in a more accurate, predictive model, as it will in this case more closely mimic the actual internal circuitry.

4. 1. 2. Definition of Setup Time

To model sequential cells with reasonable accuracy, one needs to determine not only delays, but also timing constraints exhibited by the cells, such as:

- setup and hold times
- recovery and / or removal times, namely the amount of time which has to elapse before and after the active clock edge until an asynchronous signal is deactivated (e.g. the recovery time of a reset signal is the amount of time by which an asynchronous reset signal must be deasserted before clock).
- minimal and maximal clock pulse widths.

It may be noted that not all the above signals are equally important to characterize. Hold times are important only if they are particularly lengthy. Most cells for low-power applications are constructed using static CMOS technology, and their hold times tend to be short or even negative, meaning that the flip-flops or latches may still lock the correct value even if a new input transition into an unintended value already occurs before the clock comes. As such, their hold times do not really need to be characterized. Asynchronous constraints, such as recovery and removal times, do not really need to be characterized very accurately for most situations, since generally when such signals are used in a circuit, their use come with generous amount of time given to the circuit to come out from, for example, reset state. Further, static CMOS circuits can hold their output indefinitely in the absence of clock, and hence have no maximal clock width requirements. Also, the minimal clock widths tend to be the same order with gate delays, and hence unimportant since the maximum frequency attainable by the circuit tends to be determined by critical paths, which generally far exceed minimal clock period attainable in length.

With these considerations in mind, this work will limit timing characterization of sequential constraints to setup time characterization. Consequently only setup time will be defined more precisely.

Setup time *is defined as the amount of time before the latching clock edge in which an input signal has to already reaches its expected value, so that the output signal will reach the expected logical value within a specific delay*. This delay must not change considerably if the input transition is made to occur any earlier. The following figure illustrates this definition

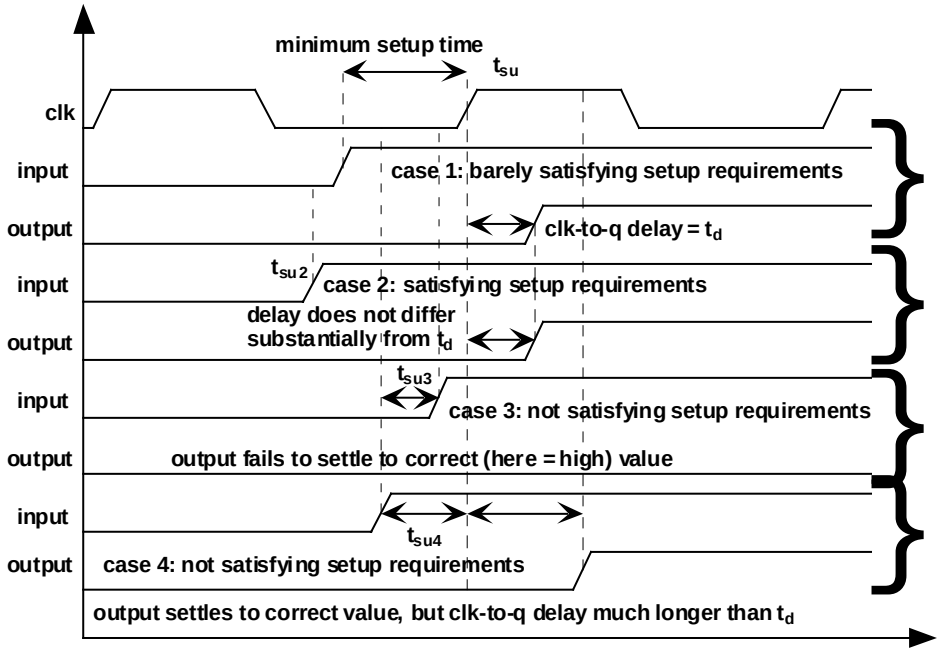


Figure 4.6. Input Rise Setup Time Definition for A Rising-Edge Triggered Flip-flop

In Figure 4.6, the definition of setup time for rising input (based on 50%-to-50% delay) is illustrated. As defined by **case 4**, for an input signal to be called satisfying setup requirements, it is not sufficient that the output value should converge to the correct value. Instead, the clock-to-output delay (or the input-to-output delay for a latch) must not change substantially if the input is made to come any earlier (compare **case 1** and **case 2**). Note also that for use with Synopsys, setup time is to be defined with respect to the latching edge of the clock. For a latch, the latching edge is the edge which ends the active level, as illustrated in the following figure.

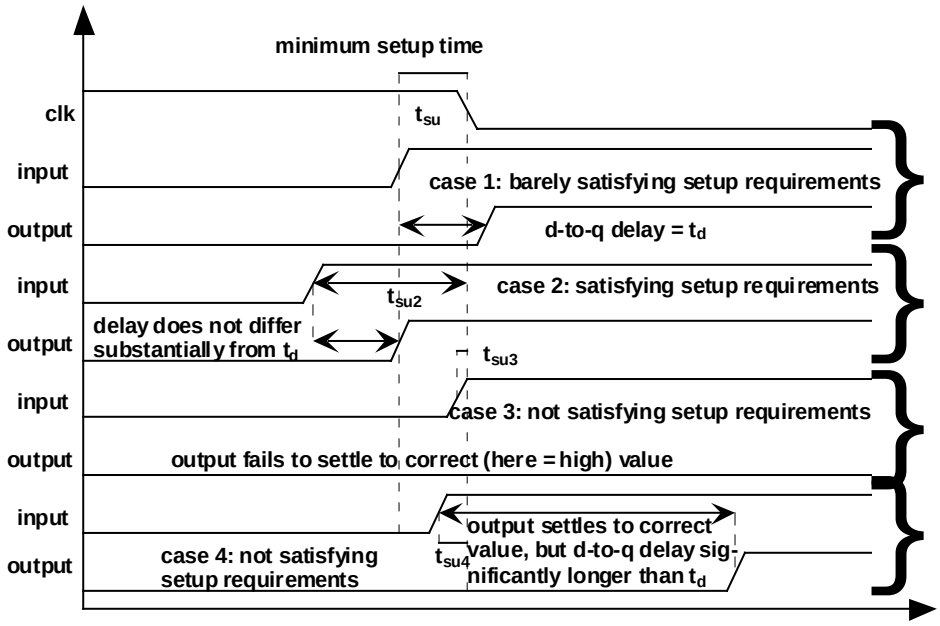


Figure 4.7. Input Rise Setup Time Definition for A High-Active Latch

As illustrated in Figure 4.7, the reference edge for a high-active latch is the falling edge, since it is the falling edge which locks the value of the output.

4. 2. Proposed Timing and Capacitance Characterization Method

4. 2. 1. Input Capacitance Measurement

4. 2. 1. 1. Basic Approach

Input capacitance values are used by both power estimation tools (to measure dynamic power dissipation) and for delay (speed) estimation tools. The capacitance is related by the voltage across a capacitor and the charge accumulated as below.

$\mathbf{C = Q/V} = I\,\Delta t/V$

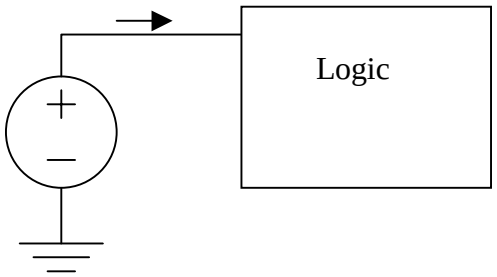


Figure 4.8. Measurement of Input Capacitance

In our case, where we are to find the input capacitance of input pins logic cells, this could be performed by applying a stimulus (a 0V-to-VDD rising pulse or a VDD-to-0V falling pulse) to the input pin and then measuring the amount of charge to have been flown into (or out from) the pin. If the period of the charge accumulation is sufficiently long, the capacitance can be obtained as the amount of charge divided by the magnitude of the stimulus VDD. ".measure" and "integral" commands of HSPICE are usually used for the measurement.

The amount of charge flown into a pin depends on the status of the other inputs as well as the output. This implies that the input capacitance depends on the status of the other inputs and the output. An accurate model may be to provide three cases, best, average, and worst, through simulation of all possible cases, but it is quite laborious. In the following, we measure input capacitances by consider one or a fest specific cases, which are intended to capture the largest (worst) capacitance.

4. 2. 1. 2. Input Capacitance Measurement for Combinational Cells

The stimulus of an input under consideration is chosen to cause a rising transition on the output. The waveforms in Figure 4.9 illustrate the measurement of input pin capacitance for a 2-input OR gate. The input capacitance of pin ip1 is measured at the rising transition at 1 ns, and that for pin ip2 is at the rising transition at 5 ns.

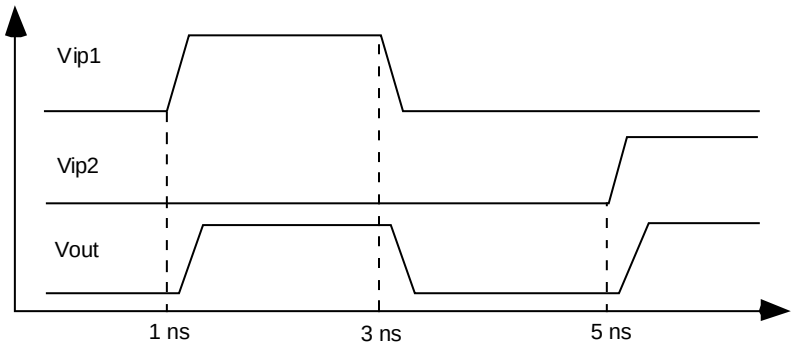


Figure 4.9. Waveforms for the measurement of the input capacitance for OR2

File `hor2_cap.sp` illustrates the measurement procedure.

```
.inc hor2.sp
* The SPICE model of the "hor2" OR gate

.param vspl='3.3v'
vdd dd gnd vspl
xhivtested dd ip1 ip2 op hor2

vin1 ip1 gnd pwl (0ns 0 1ns 0 1.01ns vspl 3ns vspl 3.01ns 0)
* The stimulus source for "ip1" input pin ...
* Probably a rise/fall time of 0.01ns is way too fast for most purpose, but
* for Cin determination it does not do any harm. Actually values in 0.1-0.3
* ns range for fall time and 0.2-0.4 ns for rise time are more realistic.

vin2 ip2 gnd pwl (0ns 0 5ns 0 5.01ns vspl)
* and the one for "ip2" input pin.

.measure qin1 integral i(vin1) from 1ns to 2.5ns
* That is, (charge accumulated in ip1) = (current from vin1) dt.
* See also chapter 4 of HSPICE manual (p. 4-19) if more info on the syntax of
* .measure and integral statements / functions are desired.
* Note also that although the input transition takes only 10 ps (ends at
* 1.01ns), the integration has to continue until practically all output
* transitions caused by the particular input transition have been completed.

.measure cin1 param='abs(qin1/vspl)'
* C = Q/V. And the magnitude of the stimulus was vspl.

* And now, we do the same thing on the "ip2" input pin.
.measure qin2 integral i(vin2) from 5ns to 6.5ns
.measure cin2 param='abs(qin2/vspl)'

.tran 0.005ns 6.5ns

.op
```

```
.save
.option post nomod accurate
.option converge=1 gmindc=1e-12
* The above statement is often neccessary - although not always. It helps
* avoiding the dreaded "no convergence in dc analysis" SPICE error message.
.end

* (END OF SPICE FILE)
```

As a result of simulating the model (hspice cdethor2.sp > cdethor2.lis) the .mt# output file generated by the .measure statements (cdethor2.mt0) contains the following info (which could also be found in a different format in cdethor2.lis)

```
$DATA1 SOURCE='HSPICE' VERSION='98.2 '
.TITLE 'hor cin determination'
 qin1 cin1 qin2 cin2 temper alter#
 -1.365e-13  4.136e-14 -1.389e-13  4.209e-14   25.0000    1.0000
```

As it turns out, we obtain Cip1 = 41.36fF and Cip2 = 42.09fF.

4. 2. 1. 3. Input Capacitance Measurement for Sequential Cells

Due to potential large differences in charge (and hence capacitance) during the rising and falling output transitions, it is prudent to consider both cases for sequential cells and average them. In general, it is a good idea to consider all possible cases for sequential cells and take the average value.

The waveform in Figure 4.10 illustrates the measurement of input D and clk for a D flip-flop. The input capacitance of D is measured at the output rising transition at 4.5 ns and at the output. The capacitance for the clock input is measured for rising clock for four cases:

i) Q changes from 0 to 1 (at 6ns)

ii) Q remains high (at 10 ns)

iii) Q changes from 1 to 0 (at 14 ns)

iv) Q changes remains at low (at 18 ns)

And the average of those four values are taken as our estimate.

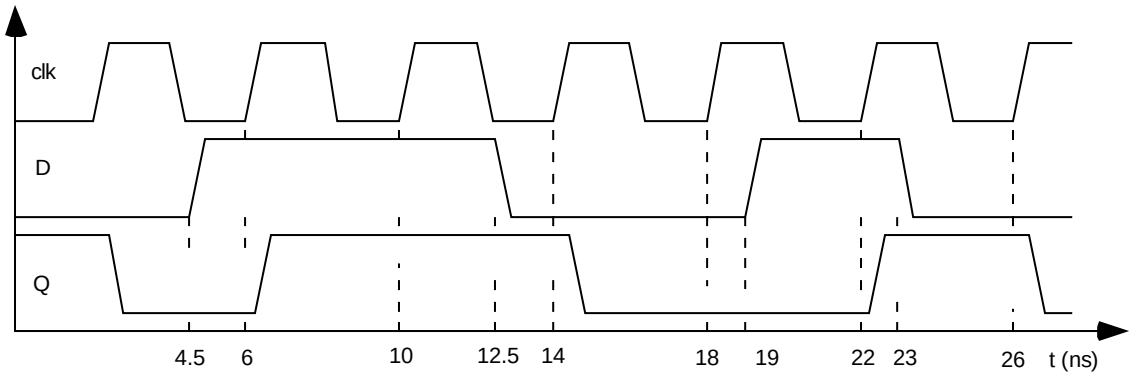


Figure 4.10. Waveforms for the measurement of the input capacitance for D flip-flop

For estimation of input capacitance for the D input, four cases are considered:

i) Rising D input with clock low (at 4.5 ns)

ii) Falling D input with clock low (at 12.5 ns)

iii) Rising D input with clock high (at 19 ns)

iv) Falling D input with clock low (at 23 ns)

The results of (i) and (ii) are averaged to obtain Cin(D) estimate for clock = low case, while the results of (iii) and (iv) are also averaged to obtain Cin(D) for clock = high, and the estimate is the larger of those two. **This procedure is to be followed for all input signals other than clock.**

In general, for combinational cells, it is prudent to consider all cases. File hdpq_cap.sp illustrates the measurement procedure. Both the rise time tr and fall time tf = 0.4ns for all signals.

```
hdpq Cin determination

.inc hdpq.sp  ** The netlist for “hdpq” is in another file named “hdpq.sp”

.param vspl='3.3v'
vdd dd gnd vspl
xhdpqtested dd d0 ck q hdpq

** clock and (if present) other stimuli **

vck ck gnd pulse (0v vspl 2ns 0.4ns 0.4ns 1.6ns 4ns)
vd0 d0 gnd pwl (0ns 0 4.5ns 0 4.9ns vspl 12.5ns vspl 12.9ns 0 19ns 0 19.4ns
+ vspl 23ns vspl 23.4ns 0)
```

```
** Measuring Cin for D input pin ...
.measure qd0rise integral i(vd0) from 4.5ns to 6ns
.measure qd0fall integral i(vd0) from 12.5ns to 14ns
** Measure Q accumulated for both rising and falling d0 for clk = low ..
.measure qd0rise_ckh integral i(vd0) from 19ns to 20ns
.measure qd0fall_ckh integral i(vd0) from 23ns to 24ns
** and for clk = high ...
.measure cd0rise param='abs(qd0rise/vspl)'
.measure cd0fall param='abs(qd0fall/vspl)'
.measure cd0rise_ckh param='abs(qd0rise_ckh/vspl)'
.measure cd0fall_ckh param='abs(qd0fall_ckh/vspl)'
** and make capacitance estimates accordingly ...
.measure cd0_ckl param='0.5*(cd0rise+cd0fall)'
.measure cd0_ckh param='0.5*(cd0rise_ckh+cd0fall_ckh)'
.measure cd0 param=’max(cd0_ckl, cd0_ckh)’
** and take the maximum of clk-low and clk-high average as our final
** estimate.

.measure qck_qrise integral i(vck) from 6ns to 8ns
.measure qck_qfall integral i(vck) from 14ns to 16ns
.measure qck_qhigh integral i(vck) from 10ns to 12ns
.measure qck_qlow integral i(vck) from 18ns to 20ns

.measure cck_qrise param='abs(qck_qrise/vspl)'
** Cin(ck) for active edge which causes rising output ..
.measure cck_qfall param='abs(qck_qfall/vspl)'
** and which causes falling output transition ...
.measure cck_qhigh param='abs(qck_qhigh/vspl)'
** and no output transition, high output ..
.measure cck_qlow param='abs(qck_qlow/vspl)'
** and low output, then take average.
.measure cck param='0.25*(cck_qrise+cck_qfall+cck_qhigh+cck_qlow)'

.param tsimlength='28ns'

.tran 0.05ns tsimlength

.op
.save
.option post nomod accurate
.option converge=1 gmindc=1e-12
.end
```

The .measure output (hdpq_cap.mt0) was as follows:

```
$DATA1 SOURCE='HSPICE' VERSION='1999.2'
.TITLE 'hdpq cin determination'
 qd0rise qd0fall qd0rise_ckh qd0fall_ckh cd0rise cd0fall cd0rise_ckh
 cd0fall_ckh cd0_ckl cd0_ckh cd0 qck_qrise qck_qfall qck_qhigh qck_qlow
 cck_qrise cck_qfall cck_qhigh cck_qlow cck temper alter#
 -1.361e-13  1.326e-13 -1.284e-13  1.261e-13  4.124e-14  4.017e-14  3.892e-14
  3.822e-14  4.071e-14  3.857e-14  4.071e-14 -3.790e-13 -2.795e-13 -3.298e-13
```

```
 -3.493e-13  1.148e-13  8.471e-14  9.993e-14  1.058e-13  1.013e-13   25.0000
    1.0000
```

So, the estimated values of Cin(d0) and Cin(ck) are 40.71fF and 101.30fF, respectively.

4. 2. 2. Non-tristate Delay Characterization

4. 2. 2. 1. Intrinsic Delay Measurement

As previously mentioned, *the intrinsic delay of a cell is defined as the propagation delay of the cell without load, when it is driven by another identical loadless cell*. The basic circuit is as shown in Figure 4.1.

For example, to measure the intrinsic delay of an inverter using SPICE, the circuit in Figure 4.11 may be used.

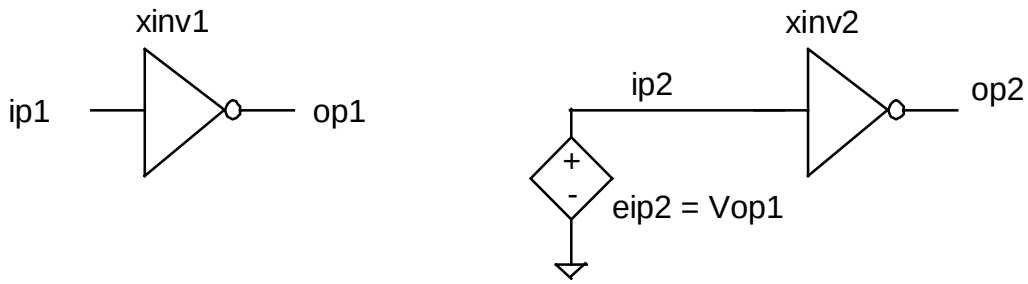


Figure 4.11. Measurement of intrinsic delays for inverters

In this example, a chain of only two inverters of "hiv" is used. The spice model for the instantiation of the inverters is as follows.

```
xinv1 dd ip1 op1 hiv
eip2 ip2 gnd op1 gnd 1
xinv2 dd ip2 op2 hiv
```

(The prefix "e" is for voltage-controlled (dependent) voltage sources, and its syntax is:
<exx> <output-nodes> <controlling-nodes> <scale factor>.)

Mathematically, the purpose of the first inverter xiv1 is to provide a well-defined value of transition delay of zero, so that the delay of the second stage could be claimed as intrinsic delay (i.e. does not include any slope delay – see eqs. 4.6-4.7). Alternatively, it could also be stated that *the purpose of the first inverter is to shape the waveform* (here op1) *so that it has the typical shape of a waveform with a transition delay of zero*. For instance, the waveform should have no sharp corners. It follows that the longer the chain, the better, but simulation speed may suffer. If only two elements are used (i.e. the waveform shaper consists of only one gate), it is recommended that the original stimulus ip1 must have rise and fall times not too far different from rise and fall times of an unloaded cell – this is in fact always recommendable regardless of chain length. For example, in most of our works, 0%-to-100% rise and fall times of 0.2ns and 0.1ns are almost always used, as the unloaded inverters usually have 10%-to-90% rise and fall times of around 0.18ns and 0.11ns, respectively. These values could be obtained by first simulating a ring oscillator of inverters whose stages are connected via dependent sources as in Figure 4.11. It is not recommendable, however, to use ring oscillators for intrinsic delay measurements of all cells for one reason, namely that mathematically it is not easy to extend the analysis of a ring oscillator to a form which will easily yield estimates of output resistances or slope sensitivities.

In all characterizations to have been undertaken in this work, the chain used for intrinsic delay determination has consisted of three cells in series, i.e. the waveform shaper consists of two cells in series. The main reason is convenience. Since many of the cells characterized were inverting, connecting two of them will create a noninverting wave shaper, which is easier to analyze using pencil and paper. Further, for the sake of consistency, the same configuration is used always, even for noninverting cells. Examples of simulation files so constructed will appear later in this chapter.

Returning to Figure 4.11, the voltage-controlled voltage source eip1 generates a voltage between nodes ip2 and ground, whose value is 1× (the same as) the voltage between the nodes op1 and grond. In other words, although the input ip2 of xinv2 is not electrically connected to the output op1 of xinv1, v(ip2) is the same as v(op1).

Assuming the second inverter (xinv2) does not drive any load, **intrinsic_rise** and **intrinsic_fall** delays could be found by measuring **tplh** and **tphl** for xinv2 as shown in Figure 4.12. Note that **all delays are for inverter 2**, not for inverter 1.

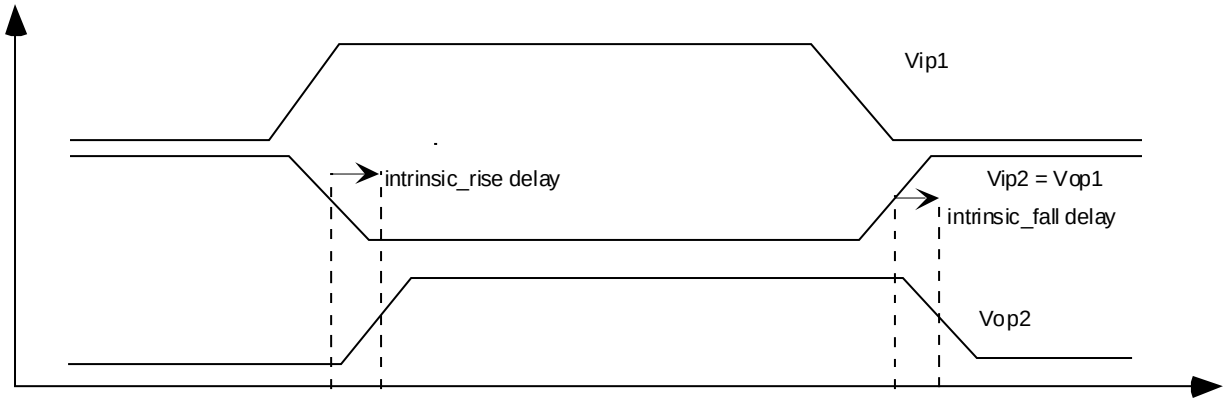


Figure 4.12. Waveforms used for inverter intrinsic delay determination

.measure statements for the SPICE model look like the following one (with vspl = the power supply voltage). Note that the propagation delay measures 50%-to-50% delays.

```
.measure intrinsic_rise trig v(ip2) val='0.5*vspl' fall=1 targ v(op2)
+ val='0.5*vspl' rise=1
* Measuring time elapsed between v(ip2) reaching (0.5*vspl) on its first rise
* and v(op2) reaching (0.5*vspl) on its first fall.
.measure intrinsic_fall trig v(ip2) val='0.5*vspl' rise=1 targ v(op2)
+ val='0.5*vspl' fall=1
```

4. 2. 2. 2. Transition Delay and Output Resistance Measurement

To recall our definition, transition delay is defined as the additional delay (in addition to intrinsic delays) of a cell driving a capacitive load, but which is driven by another identical loadless cell. Therefore, to find transition delay, a setup as in Figure 4.2 is used to compare the delay of a loaded cell with the delay of an unloaded cell driven by the same waveform; the difference is transition delay. Further, output resistance is simply this extra delay divided by load capacitance (*cf*. Eqs. 4.2, 4.4, 4.5). Or, mathematically:

rise transition delay = loaded rise delay – unloaded rise delay (4.10)

fall transition delay = loaded fall delay – unloaded fall delay (4.11)

rise output resistance = rise transition delay / load capacitance (4.12)

fall output resistance = fall transition delay / load capacitance (4.13)

An example of circuit for transition delay / output resistance measurement is the one for inverters shown in Figure 4.13. A circuit in which an inverter driving two inverters in parallel, one with a load capacitor and one without a load capacitor. The length of the chain is less critical here than is the case for intrinsic delay determination. At any case, to ensure consistency, the same chain should be used. In the actual simulations to have been undertaken so far, a chain length of 3 cells have been used, as previously mentioned in 4. 2. 2. 1.

The propagation and transition delays of the inverter with the load are shown in Figure 4.14 (next page). The transition delay for the inverter xinv2L in is obtained as (the propagation delay of xinv2L – the propagation delay of xinv2).

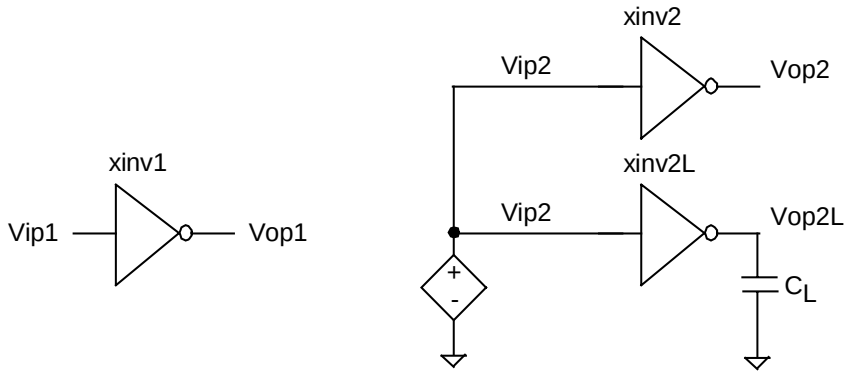


Figure 4.13. Measurement of transition delay for inverters

The procedure for measuring output resistance is as follows. After the intrinsic delays of a cell are measured first, propagation delays are measured for a known load capacitance. Then transition delays are obtained from the two delays. The output resistance is calculated as the transition delay divided by the load capacitance.

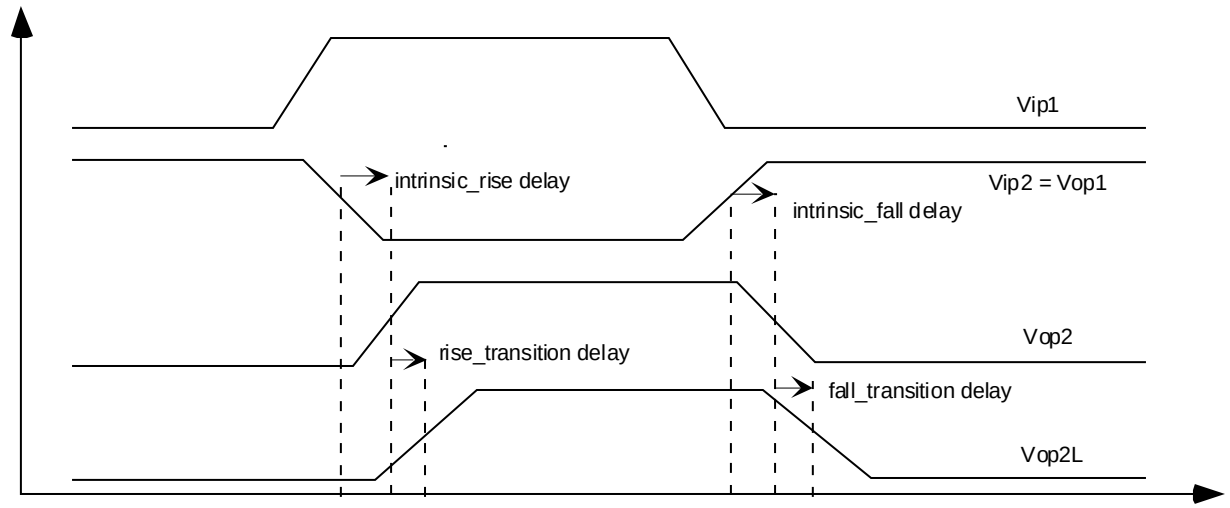


Figure 4.14. Waveforms used for circuit in Figure 4.13.

A SPICE model to compute load resistance for inverters is shown below. A load capacitance is set to 40 fF in the following simulation, as it is roughly the same order of the input capacitance of an inverter for 0.5 μm technology.

```
xinv2L dd ip2L op2L hiv
eip2L ip2L gnd op1 gnd 1
.param loaderc='40fF'
cload op2L gnd loaderc

.measure tr_rise trig v(ip2L) val='0.5*vspl' fall=1 targ v(op2L)
+ val='0.5*vspl' rise=1
.measure tr_fall trig v(ip2b) val='0.5*vspl' rise=1 targ v(op2b)
+ val='0.5*vspl' fall=1
.measure dt_rise param='tr_rise - intrinsic_rise'
.measure dt_fall param='tr_fall - intrinsic_fall'
.measure rise_resistance param='dt_rise/loaderc'
.measure fall_resistance param='dt_fall/loaderc'
```

In general, a value of 1 – 2.5 × the input capacitance of the inverter would be a good choice, as it would be in the same order with the load usually driven by a typical cell. Note, however, that the smaller the minimum feature size of the technology, the higher should the ratio of C_{load} / (C_{in} of inverter) be, as wire capacitances tend to be quite dominant in deep submicron technologies, and hence the relative load seen by the output pin (relative to C_{in} of an inverter) increases, even if the number of cell being driven does not increase.

For the work performed so far, the following values had been used. ***(Note that for combinational cells, the same values should also be used for slope delay characterization, which will be discussed in the next section)***

Cell type	Load Capacitance (fF)
0.5um, combinational, standard drive	40
0.5um, combinational, high drive	150
0.5um, sequential	150
0.35um, combinational, drive strength 1	40
0.35um, combinational, drive strength 2	75
0.35um, combinational, drive strength 4	150
0.35um, combinational, drive strength 8	250
0.35um, combinational, drive strength 12	400
0.35um, combinational, drive strength 16	600
0.35um, combinational, drive strength 20	1000
0.35um, sequential, drive strength 2	100
0.35um, sequential, drive strength 4	150

4. 2. 2. 3. Slope Sensitivity Measurement

As previously defined, slope delay is an extra delay of a loadless cell which is driven by an identical cell with transition delay. The driving cell drives a capacitive load (and so it exhibits transition delay), and hence the output slope of the driving cell is less steep than the one without a load capacitance. Hence, returning to Figure 4.3, slope sensitivity could be found using the following steps:

- measurement of intrinsic delay
- measurement of delay of cell exhibiting slope delay; slope delay is found as the difference
- slope sensitivity is obtained as (slope delay) / (transition delay of driving waveform)

As an example, Figures 4.15 – 4.16 illustrate the measurement of slope delay and slope sensitivity of the inverter. Here the inverter whose slope delay is measured is xinv3S. Inverter xinv2L (with a load capacitor) drives xinv3s through a dependent voltage source. The difference of the propagation delay between xinv2 and xinv3S is the slope delay of inverter xinv3S. The

capacitances used here are the same value of capacitance as used for output resistance determination.

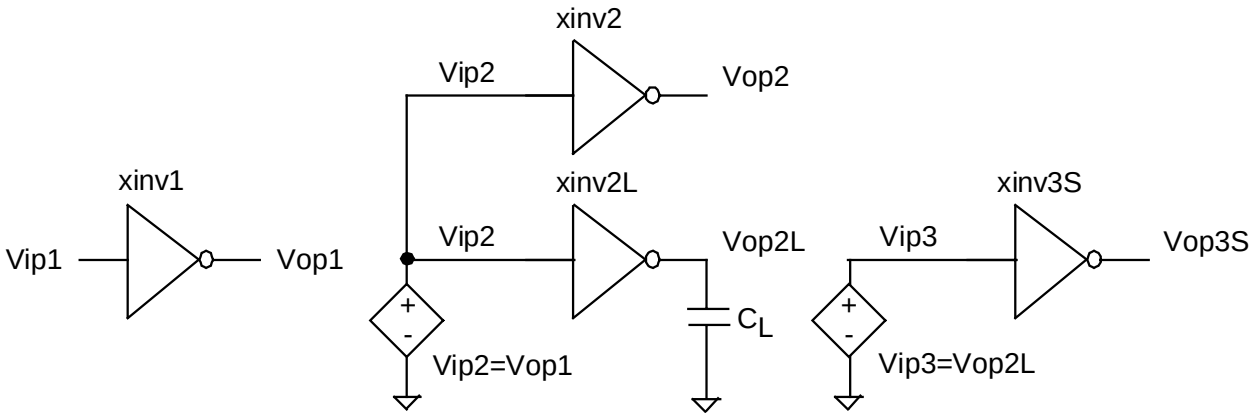


Figure 4.15. Measurement of slope delay for inverters

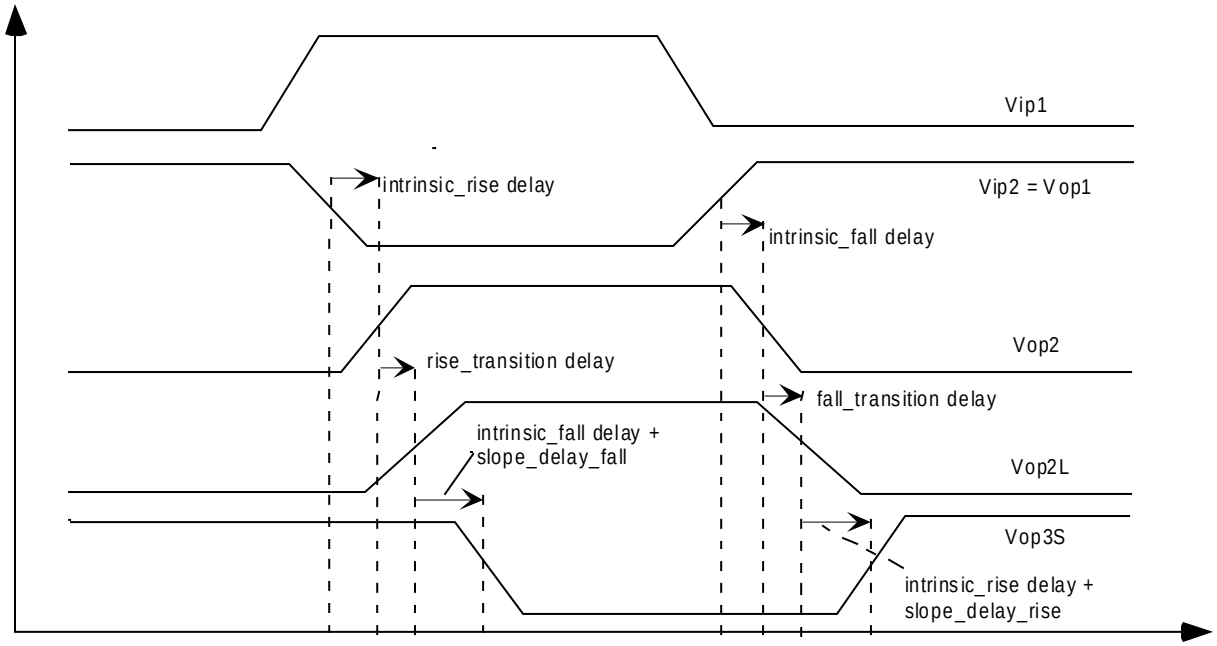


Figure 4.16. Slope delays of inverters

A slope parameter, slope_rise, for the inverters is obtained as follows. The propagation delay tplh of xinv3S and tplh of xinv2 (which is the intrinsic_rise delay) are measured. The difference of the two delays is the slope_delay_rise of xinv3S. The fall transition delay of xinv2L is obtained from tphl of xinv2L and tphl of xinv2. The slop_rise parameter is slope_delay_rise of xinv3S divided by fall transition delay of xinv2L. Similarly, slope_fall parameter can be obtained. Or, mathematically, for the inverter:

slope delay rise = rise delay of xinv3S – intrinsic rise delay (4.14)

slope delay fall = fall delay of xinv3S – intrinsic fall delay (4.15)

rise slope sensitivity = slope delay rise / fall transition delay of xinv2L (4.16)

fall slope sensitivity = slope delay fall / rise transition delay of xinv2L (4.17)

Note that the term rise (or fall) slope sensitivity refers to the rising (or falling) output of the inverter being observed (xinv3S) rather than its input. The following lines were added in the previous SPICE model to compute slope_delay_rise parameter:

```
eslopes sip gnd op2L gnd 1
xhivslopes dd sip op3S hiv

.measure ss_rise trig v(op2L) val='0.5*vspl' fall=1 targ v(op3S)
+ val='0.5*vspl' rise=1
.measure ss_fall trig v(op2L) val='0.5*vspl' rise=1 targ v(op3S)
+ val='0.5*vspl' fall=1
.measure st_rise param='sl_rise - intrinsic_rise'
.measure sd_fall param='sl_fall - intrinsic_fall'
.measure slope_rise param='st_rise/dt_fall'
* Note that we have an inversion here
.measure slope_fall param='st_fall/dt_rise'
```

4. 2. 2. 4. SPICE Examples of Capacitance and Delay Characterization

Here two examples of SPICE files to actually have been simulated are presented: one is for a 2-input NAND gate, and the other for a simple D flip-flop. For a complete delay characterization, it is necessary to write a SPICE model to measure the following parameters for a cell:

– input capacitances of all input pins

– intrinsic delays
– output resistance (rise_resistance and fall_resistance)
– slope sensitivities (slope_rise and slope_fall)

4. 2. 2. 4. 1. Example for a 2-input NAND gate

This is a SPICE model (file name: hnd2_tim.sp) used to characterize a 0.5μm, 2-input NAND gate. The last section is to test the accuracy of the characterization model. The parameters extracted from simulation are used to predict the speed of two 11-element mini-ring-oscillators would run. The result is compared with SPICE models of the two ring oscillator. The result is reasonably accurate as discussed later.

```
* hnd2_tim.sp

.inc hnd2.sp

.TEMP    25.0000

** Power Supply
.param vspl='3.3v'
vdd dd gnd vspl

** Input Capacitance Determination  **
xhnd2cindet dd ipcdet1 ipcdet2 opcdet2 hnd2
vipcdet1 ipcdet1 gnd pwl (0ns 0 1ns 0 1.2ns vspl)
vipcdet2 ipcdet2 gnd pwl (0ns vspl 3ns vspl 3.1ns 0 5ns 0 5.2ns vspl)
.measure qincdet1 integral i(vipcdet1) from=1ns to=2.5ns
.measure cip1est param='abs(qincdet1/vspl)'
.measure qincdet2 integral i(vipcdet2) from=5ns to=6.5ns
.measure cip2est param='abs(qincdet2/vspl)'

** Intrinsic Rise / Fall Determination **
xbufnhd1a dd ip1a dd ip1n loadfreehnd2
xbufhnd2a dd dd ip2a ip2n loadfreehnd2
xbufnhd1b dd ip1n dd ip1 loadfreehnd2
xbufhnd2b dd dd ip2n ip2 loadfreehnd2
xhnd2tested dd ip1 ip2 op loadfreehnd2
vip1 ip1a gnd pwl(0ns vspl 2ns vspl 2.1ns 0 4ns 0 4.2ns vspl)
vip2 ip2a gnd pwl(0ns vspl 6ns vspl 6.1ns 0 8ns 0 8.2ns vspl)
.measure intrise_ip1op trig v(ip1) val='0.5*vspl' fall=1 targ v(op)
+ val='0.5*vspl' rise=1
.measure intfall_ip1op trig v(ip1) val='0.5*vspl' rise=1 targ v(op)
+ val='0.5*vspl' fall=1
.measure intrise_ip2op trig v(ip2) val='0.5*vspl' fall=1 targ v(op)
+ val='0.5*vspl' rise=2
.measure intfall_ip2op trig v(ip2) val='0.5*vspl' rise=1 targ v(op)
```

```
+ val='0.5*vspl' fall=2

** Output Resistance Determination **
xhnd2rout dd ip1 ip2 oprout loadfreehnd2
.param loaderc_rout='40fF'
cloaderc_rout oprout gnd loaderc_rout
.measure rotrise_ip1op trig v(ip1) val='0.5*vspl' fall=1 targ v(oprout)
+ val='0.5*vspl' rise=1
.measure rotfall_ip1op trig v(ip1) val='0.5*vspl' rise=1 targ v(oprout)
+ val='0.5*vspl' fall=1
.measure rotrise_ip2op trig v(ip2) val='0.5*vspl' fall=1 targ v(oprout)
+ val='0.5*vspl' rise=2
.measure rotfall_ip2op trig v(ip2) val='0.5*vspl' rise=1 targ v(oprout)
+ val='0.5*vspl' fall=2
.measure dtrise_ip1op param='rotrise_ip1op-intrise_ip1op'
.measure dtfall_ip1op param='rotfall_ip1op-intfall_ip1op'
.measure dtrise_ip2op param='rotrise_ip2op-intrise_ip2op'
.measure dtfall_ip2op param='rotfall_ip2op-intfall_ip2op'
.measure rise_r_ip1op param='dtrise_ip1op/loaderc_rout'
.measure fall_r_ip1op param='dtfall_ip1op/loaderc_rout'
.measure rise_r_ip2op param='dtrise_ip2op/loaderc_rout'
.measure fall_r_ip2op param='dtfall_ip2op/loaderc_rout'

** Slope Sensitivity Measurement **
xbufnhd1c dd ip1n dd ip1sl loadfreehnd2
xbufhnd2c dd dd ip2n ip2sl loadfreehnd2
.param loaderc_slsens='40fF'
csens1 ip1sl gnd loaderc_slsens
csens2 ip2sl gnd loaderc_slsens
xhnd2slsens dd ip1sl ip2sl opsl loadfreehnd2
.measure tinr_ip1 trig v(ip1n) val='0.5*vspl' fall=1 targ v(ip1)
val='0.5*vspl'
+ rise=1
.measure tinf_ip1 trig v(ip1n) val='0.5*vspl' rise=1 targ v(ip1)
val='0.5*vspl'
+ fall=1
.measure tinr_ip2 trig v(ip2n) val='0.5*vspl' fall=1 targ v(ip2)
val='0.5*vspl'
+ rise=1
.measure tinf_ip2 trig v(ip2n) val='0.5*vspl' rise=1 targ v(ip2)
val='0.5*vspl'
+ fall=1
.measure tdtr_ip1 trig v(ip1n) val='0.5*vspl' fall=1 targ v(ip1sl)
+ val='0.5*vspl' rise=1
.measure tdtf_ip1 trig v(ip1n) val='0.5*vspl' rise=1 targ v(ip1sl)
+ val='0.5*vspl' fall=1
.measure tdtr_ip2 trig v(ip2n) val='0.5*vspl' fall=1 targ v(ip2sl)
+ val='0.5*vspl' rise=1
.measure tdtf_ip2 trig v(ip2n) val='0.5*vspl' rise=1 targ v(ip2sl)
+ val='0.5*vspl' fall=1
.measure dtr_ip1 param='tdtr_ip1-tinr_ip1'
.measure dtf_ip1 param='tdtf_ip1-tinf_ip1'
.measure dtr_ip2 param='tdtr_ip2-tinr_ip2'
.measure dtf_ip2 param='tdtf_ip2-tinf_ip2'
.measure sltrise_ip1op trig v(ip1sl) val='0.5*vspl' fall=1 targ v(opsl)
+ val='0.5*vspl' rise=1
.measure sltfall_ip1op trig v(ip1sl) val='0.5*vspl' rise=1 targ v(opsl)
```

```
+ val='0.5*vspl' fall=1
.measure sltrise_ip2op trig v(ip2sl) val='0.5*vspl' fall=1 targ v(opsl)
+ val='0.5*vspl' rise=2
.measure sltfall_ip2op trig v(ip2sl) val='0.5*vspl' rise=1 targ v(opsl)
+ val='0.5*vspl' fall=2
.measure dslrise_ip1op param='sltrise_ip1op-intrise_ip1op'
.measure dslfall_ip1op param='sltfall_ip1op-intfall_ip1op'
.measure dslrise_ip2op param='sltrise_ip2op-intrise_ip2op'
.measure dslfall_ip2op param='sltfall_ip2op-intfall_ip2op'
.measure slope_rise_ip1op param='dslrise_ip1op/dtf_ip1'
.measure slope_fall_ip1op param='dslfall_ip1op/dtr_ip1'
.measure slope_rise_ip2op param='dslrise_ip2op/dtf_ip2'
.measure slope_fall_ip2op param='dslfall_ip2op/dtr_ip2'

** PREDICTION TESTING - RING OSCILLATOR **

* RING OSCILLATOR 1 -  WITH ip2 TIED TO VDD, 3 LOADS PER ELEMENT *
.subckt hnd2ip2c dd ip op
xhnd2 dd ip dd op hnd2
xload1 dd op dd opx1 hnd2
xload2 dd op dd opx2 hnd2
.ends hnd2ip2c
.ic v(r1ip01)=0v
xel1_01 dd r1ip01 r1ip02 hnd2ip2c
xel1_02 dd r1ip02 r1ip03 hnd2ip2c
xel1_03 dd r1ip03 r1ip04 hnd2ip2c
xel1_04 dd r1ip04 r1ip05 hnd2ip2c
xel1_05 dd r1ip05 r1ip06 hnd2ip2c
xel1_06 dd r1ip06 r1ip07 hnd2ip2c
xel1_07 dd r1ip07 r1ip08 hnd2ip2c
xel1_08 dd r1ip08 r1ip09 hnd2ip2c
xel1_09 dd r1ip09 r1ip10 hnd2ip2c
xel1_10 dd r1ip10 r1ip11 hnd2ip2c
xel1_11 dd r1ip11 r1ip01 hnd2ip2c
.measure period1_act trig v(r1ip02) val='0.5*vspl' fall=1 targ v(r1ip02)
+ val='0.5*vspl' fall=2
.measure tplh1_act trig v(r1ip02) val='0.5*vspl' fall=1 targ v(r1ip03)
+ val='0.5*vspl' rise=1
.measure tphl1_act trig v(r1ip02) val='0.5*vspl' rise=1 targ v(r1ip03)
+ val='0.5*vspl' fall=1
.measure cip1 param='cip1est'
.measure dtr1_pred param='3*cip1*rise_r_ip1op'
.measure dtf1_pred param='3*cip1*fall_r_ip1op'
.measure dtsr1_pred param='slope_rise_ip1op*dtf1_pred'
.measure dtsf1_pred param='slope_fall_ip1op*dtr1_pred'
.measure tplh1_pred param='intrise_ip1op+dtr1_pred+dtsr1_pred'
.measure tphl1_pred param='intfall_ip1op+dtf1_pred+dtsf1_pred'
.measure period1_pred param='11*(tplh1_pred+tphl1_pred)'
.measure tplh1_err param='100*abs((tplh1_pred-tplh1_act)/tplh1_act)'
.measure tphl1_err param='100*abs((tphl1_pred-tphl1_act)/tphl1_act)'
.measure period1_err param='100*abs((period1_pred-period1_act)/period1_act)'

* RING OSCILLATOR 2 -  WITH ip1 TIED TO VDD, 3 LOADS PER ELEMENT *
.subckt hnd2ip1c dd ip op
xhnd2 dd dd ip op hnd2
xload1 dd dd op opx1 hnd2
xload2 dd dd op opx2 hnd2
```

```
.ends hnd2ip1c
.ic v(r2ip01)=0v
xel2_01 dd r2ip01 r2ip02 hnd2ip1c
xel2_02 dd r2ip02 r2ip03 hnd2ip1c
xel2_03 dd r2ip03 r2ip04 hnd2ip1c
xel2_04 dd r2ip04 r2ip05 hnd2ip1c
xel2_05 dd r2ip05 r2ip06 hnd2ip1c
xel2_06 dd r2ip06 r2ip07 hnd2ip1c
xel2_07 dd r2ip07 r2ip08 hnd2ip1c
xel2_08 dd r2ip08 r2ip09 hnd2ip1c
xel2_09 dd r2ip09 r2ip10 hnd2ip1c
xel2_10 dd r2ip10 r2ip11 hnd2ip1c
xel2_11 dd r2ip11 r2ip01 hnd2ip1c
.measure period2_act trig v(r2ip02) val='0.5*vspl' fall=1 targ v(r2ip02)
+ val='0.5*vspl' fall=2
.measure tplh2_act trig v(r2ip02) val='0.5*vspl' fall=1 targ v(r2ip03)
+ val='0.5*vspl' rise=1
.measure tphl2_act trig v(r2ip02) val='0.5*vspl' rise=1 targ v(r2ip03)
+ val='0.5*vspl' fall=1
.measure cip2 param='cip2est'
.measure dtr2_pred param='3*cip2*rise_r_ip2op'
.measure dtf2_pred param='3*cip2*fall_r_ip2op'
.measure dtsr2_pred param='slope_rise_ip2op*dtf2_pred'
.measure dtsf2_pred param='slope_fall_ip2op*dtr2_pred'
.measure tplh2_pred param='intrise_ip2op+dtr2_pred+dtsr2_pred'
.measure tphl2_pred param='intfall_ip2op+dtf2_pred+dtsf2_pred'
.measure period2_pred param='11*(tplh2_pred+tphl2_pred)'
.measure tplh2_err param='100*abs((tplh2_pred-tplh2_act)/tplh2_act)'
.measure tphl2_err param='100*abs((tphl2_pred-tphl2_act)/tphl2_act)'
.measure period2_err param='100*abs((period2_pred-period2_act)/period2_act)'

** simulation directives **
.OP
.save
.OPTION  INGOLD=2 ARTIST=2 PSF=2
+        PROBE=0
.option post nomod accurate
.option converge=1 gmindc=1e-12
.tran 0.005ns 10ns

.END
```

The following command simulates the above model.

```
hspice hnd2_tim > hnd2_tim.lis
```

Output file hnd2_tim.lis contains the following lines, among others:

```
* hnd2_tim.sp
 ******  transient analysis               tnom=  25.000 temp=  25.000
******
  cip1est         =  4.2549E-14 ** The estimate for Cin(ip1) is 42.549fF ...
  cip2est         =  4.2055E-14 ** while for Cin(ip2), it is 42.055fF ...
  .
```

```
  .
  .
  intrise_ip1op   =  7.8168E-11  targ=  2.2689E-09   trig=  2.1908E-09
  intfall_ip1op   =  7.0191E-11  targ=  4.3230E-09   trig=  4.2528E-09
  intrise_ip2op   =  1.0683E-10  targ=  6.3317E-09   trig=  6.2249E-09
  intfall_ip2op   =  7.6800E-11  targ=  8.3599E-09   trig=  8.2831E-09
** So, the intrinsic delays are:
** ip1 to op: intrinsic_rise = 78.2ps, intrinsic_fall = 70.2ps
** ip2 to op: intrinsic_rise = 106.8ps, intrinsic_fall = 76.8ps
  .
  .
  .
  rise_r_ip1op    =  1.1321E+03
  fall_r_ip1op    =  1.0431E+03
  rise_r_ip2op    =  1.0675E+03
  fall_r_ip2op    =  1.0081E+03
** while the output resistances are:
** ip1 to op: rise_resistance = 1,132 Ohm, fall_resistance = 1,043 Ohm
** ip2 to op: rise_resistance = 1,068 Ohm, fall_resistance = 1,008 Ohm
  .
  .
  .
  slope_rise_ip1op=  3.1935E-01
  slope_fall_ip1op=  1.3338E-01
  slope_rise_ip2op=  3.9107E-01
  slope_fall_ip2op=  1.7129E-02
** And the slope sensitivity parameters are:
** ip1 to op: slope_rise = 0.3194, slope_fall = 0.1334
** ip2 to op: slope_rise = 0.3911, slope_fall = 0.1713
  .
  .
  .
** Just for a twist, for the ring oscillator consisting of NAND gates with
** ip2 tied to VDD, the actual period, tplh, and tphl are:
  period1_act     =  5.4665E-09  targ=  5.8328E-09   trig=  3.6632E-10
  tplh1_act       =  2.7545E-10  targ=  6.4177E-10   trig=  3.6632E-10
  tphl1_act       =  2.2128E-10  targ=  3.3479E-09   trig=  3.1266E-09
  .
  .
  .
** while the model's predictions are:
  tplh1_pred      =  2.6501E-10
  tphl1_pred      =  2.2218E-10
  period1_pred    =  5.3591E-09
  tplh1_err       =  3.7890E+00 ** for tplh - misses by 3.79%
  tphl1_err       =  4.1220E-01 ** for tphl - misses by 0.41%
  period1_err     =  1.9640E+00 ** for period - misses by 1.96%
** while for the mini-ring-oscillator with ip2 VDD'ed are:
** actual results:
  period2_act     =  5.4825E-09  targ=  5.8588E-09   trig=  3.7626E-10
  tplh2_act       =  3.0951E-10  targ=  6.8576E-10   trig=  3.7626E-10
  tphl2_act       =  1.8844E-10  targ=  3.3717E-09   trig=  3.1832E-09
  .
  .
  .
** while the model's predictions are ....
  tplh2_pred      =  2.9125E-10
```

```
tphl2_pred      =  2.0630E-10
period2_pred    =  5.4730E-09
tplh2_err       =  5.8992E+00 ** for tplh - misses by 5.9%
tphl2_err       =  9.4509E+00 ** for tphl - misses by 9.45%
period2_err     =  1.7398E-01 ** for period - misses by 0.17%
```

In general, it is very difficult to achieve an accuracy of better than 20% with the timing model. So the result is considered as reasonably accurate.

4. 2. 2. 4. 2. Example for Simple D Flip-flop

Sequential cell timing and capacitance characterization differs from the characterization of combinational cells in the following ways:

1. For a combinational cell, we have insisted that the cell should be driven (indirectly via a voltage-dependent voltage source) *by another cell of exactly the same kind*. However, since sequential cells are often driven by non-sequential cells (such as a clock driver), this requirement does not necessarily apply. For instance, for the purpose of characterizing cells from CMC (Canadian Microelectronics Consortium) library, “hiv” inverter driving a 50fF load for 0.5μm cells and “winv_1” inverter (winv_1.sp in 0.35μm directory) driving a 40fF load for 0.35μm cells.
2. For a sequential cell, some input transition may not cause any output transitions. For a simple D flip-flop, for example, we do not need to measure D → Q (or QN) propagation delay (since a logic transition on D should wait for the clock to propagate). Note, though, that for a D latch, for example, we will have to measure both D → Q and clk → Q delay parameters.
3. See also notes on the timing for the D signal for capacitance characterization (4. 2. 1. 3).

As an illustration, the following SPICE model (file name: hdpq_tim.sp) was used to characterize a 0.5μm, D flip-flop. Note that in this case, Cin(D0) is measured only for (clock = low) case; this is acceptable here as we already know (from a previous experiment) that for this particular cell, Cin(D0) is higher for the (clock = low) case than for (clock = high) case.

```
hdpq timing characterization
```

```
.inc hdpq.sp
.inc hiv.sp ** we will need inverters later on.
.param vspl='3.3v'
vdd dd gnd vspl
** Input Capacitance Measurement
xhdpqcdet dd d0cdet ckcdet qcdet hdpq
vckcdet ckcdet gnd pulse (0v vspl 0ns 0.2ns 0.1ns 1.8ns 4ns)
vd0cdet d0cdet gnd pulse (vspl 0v 2.8ns 0.1ns 0.2ns 7.9ns 16ns)
.measure qd0r integral i(vd0cdet) from=10.8ns to=12ns
.measure qd0f integral i(vd0cdet) from=2.8ns to=4ns
.measure cd0r param='abs(qd0r/vspl)'
.measure cd0f param='abs(qd0f/vspl)'
.measure cd0est param='0.5*(cd0r+cd0f)'
.measure qck_qh integral i(vckcdet) from=0ns to=2ns
.measure qck_ql integral i(vckcdet) from=8ns to=10ns
.measure qck_qrise integral i(vckcdet) from=12ns to=14ns
.measure qck_qfall integral i(vckcdet) from=4ns to=6ns
.measure cck_qh param='abs(qck_qh/vspl)'
.measure cck_ql param='abs(qck_ql/vspl)'
.measure cck_qrise param='abs(qck_qrise/vspl)'
.measure cck_qfall param='abs(qck_qfall/vspl)'
.measure cckest param='0.25*(cck_qh+cck_ql+cck_qrise+cck_qfall)'

** Subcircuit definitions
.subckt noninvbuf dd ip op    ** this is why we need hiv.sp
xinv dd ip intmd loadfreehiv
xbck dd intmd op loadfreehiv
.ends noninvbuf
.subckt nonloadhdpq dd d0 ck q
ed0 d0in gnd d0 gnd 1
eck ckin gnd ck gnd 1
xff dd d0in ckin q hdpq
.ends nonloadhdpq

** Stimulus source for next analysis stages
vck ck gnd pulse (0v vspl 0ns 0.2ns 0.1ns 7.8ns 16ns)
vd0 d0 gnd pulse (vspl 0v 12ns 0.1ns 0.2ns 15.9ns 32ns)

** Intrinsic delay determination
xbufick dd ck cki noninvbuf
xbufid0 dd d0 d0i noninvbuf
xhdpqid dd d0i cki qi nonloadhdpq
.measure int_rise trig v(cki) val='0.5*vspl' rise=3 targ v(qi) val='0.5*vspl'
+ rise=1
.measure int_fall trig v(cki) val='0.5*vspl' rise=2 targ v(qi) val='0.5*vspl'
+ fall=1

** Output resistance measurement
xhndpqro dd d0i cki qro nonloadhdpq
.param loadcro='150fF'
cloadro qro gnd loadcro
.measure load_rise trig v(cki) val='0.5*vspl' rise=3 targ v(qro)
val='0.5*vspl'
+ rise=1
.measure load_fall trig v(cki) val='0.5*vspl' rise=2 targ v(qro)
val='0.5*vspl'
```

```
+ fall=1
.measure dtr param='load_rise-int_rise'
.measure dtf param='load_fall-int_fall'
.measure rise_resistance param='dtr/loadcro'
.measure fall_resistance param='dtf/loadcro'

** Slope sensitivity measurement
.param slopec='50fF'
xbufcksl dd ck cksl noninvbuf
cckslope cksl gnd slopec
xbufd0sl dd d0 d0sl noninvbuf
cd0slope d0sl gnd slopec
xhdpqdsl dd d0sl cksl qsl nonloadhdpq
** First, measure clock transition delay ...
.measure int_ckdr trig v(xbufick.intmd) val='0.5*vspl' fall=1 targ v(cki)
+ val='0.5*vspl' rise=1
.measure int_ckdf trig v(xbufick.intmd) val='0.5*vspl' rise=1 targ v(cki)
+ val='0.5*vspl' fall=1
.measure dt_ckdr trig v(xbufcksl.intmd) val='0.5*vspl' fall=1 targ v(cksl)
+ val='0.5*vspl' rise=1
.measure dt_ckdf trig v(xbufcksl.intmd) val='0.5*vspl' rise=1 targ v(cksl)
+ val='0.5*vspl' fall=1
.measure ckdtr param='dt_ckdr-int_ckdr'
.measure ckdtf param='dt_ckdf-int_ckdf'
** then, work at the q output .. measure appropriate sort of delays.
.measure wsd_rise trig v(cksl) val='0.5*vspl' rise=3 targ v(qsl)
val='0.5*vspl'
+ rise=1
.measure wsd_fall trig v(cksl) val='0.5*vspl' rise=2 targ v(qsl)
val='0.5*vspl'
+ fall=1
.measure sdelay_rise param='wsd_rise-int_rise'
.measure sdelay_fall param='wsd_fall-int_fall'
.measure slope_rise param='sdelay_rise/ckdtr'
.measure slope_fall param='sdelay_fall/ckdtr'

** Simulation directives
.op
.option post nomod accurate
.option converge=1 gmindc=1e-12
.tran 0.025ns 40ns
.end
```

The following command simulates the above model.

```
hspice hdpq_tim > hdpq_tim.lis
```

Output file (hdpq_tim.lis) contains the following lines, among others:

```
******
hdpq timing characterization
 ******  transient analysis               tnom=  25.000 temp=  25.000
******
.
```

```
.
.
  cd0est          =  3.9773E-14    ** Cin(D0) is estimated at 39.77fF;
** note that we estimate it differently from in the previous example
** hence the differing result
.
.
  cckest          =  1.0059E-13  ** Cin(CLK) is estimated at 100.59fF
  int_rise        =  2.5200E-10  targ=  3.2454E-08   trig=  3.2202E-08
  int_fall        =  4.2435E-10  targ=  1.6626E-08   trig=  1.6202E-08

** The intrinsic CLK -> Q fall delays are estimated at 252 and 424.35
** ps, respectively

  load_rise       =  4.1422E-10  targ=  3.2616E-08   trig=  3.2202E-08
  load_fall       =  5.5105E-10  targ=  1.6753E-08   trig=  1.6202E-08

** The above values are estimates for delays with load
.
.
  rise_resistance =  1.0815E+03
  fall_resistance =  8.4467E+02
** The rise- and fall-resistances are estimated at 1081 and 845 Ohms
 int_ckdr        =  5.7231E-11  targ=  2.0220E-10   trig=  1.4496E-10
  int_ckdf        =  4.2256E-11  targ=  8.1544E-09   trig=  8.1121E-09
  dt_ckdr         =  1.0976E-10  targ=  2.5473E-10   trig=  1.4496E-10
  dt_ckdf         =  7.4978E-11  targ=  8.1871E-09   trig=  8.1121E-09
  ckdtr           =  5.2532E-11
  ckdtf           =  3.2722E-11

** The above values are the transition delays of the clock drivers

  wsd_rise        =  2.6965E-10  targ=  3.2524E-08   trig=  3.2255E-08
  wsd_fall        =  4.3692E-10  targ=  1.6692E-08   trig=  1.6255E-08
  sdelay_rise     =  1.7645E-11
  sdelay_fall     =  1.2569E-11
  slope_rise      =  3.3589E-01
  slope_fall      =  2.3926E-01

** and slope sensitivities are 0.33589 (rise) and 0.23926 (fall).
```

4. 2. 5. Setup Time Characterization Using Bisection

4. 2. 5. 1. Discussion

As previously mentioned in 4. 1. 2, setup time is the *amount of time before the latching clock edge in which an input signal has to already reaches its expected value, so that the output signal will reach the expected logical value within a specific delay*. This delay must not change

considerably if the input transition is made to occur any earlier. To be more precise, in this work a clock-to-output delay tolerance of 1% is used, although this might have been too stringent. In other words, no matter how much earlier in the clock cycle the input is made to come, the clock-to-output delay must not vary by more than 1%.

This definition already defines the task of finding the setup time as well:

- Determine the reference delay. This is obtained by performing the input transition early enough as to insure that setup constraints are satisfied. In this work, this is performed by changing the input 2ns before clock edge.
- Perform another simulation in another flip-flop of the same type, in which the input is made closer and closer to the clock edge, and yet still guarantee that the clock-to-output delay does not exceed 1.01 × reference delay. Note also that depending on the type of the flip-flop or latch, setup time may be negative (i.e., it may be permissible for the intended input transition to come after the latching clock edge).

One obvious feature of this method is that the use of repetition is inevitable. Hence, the problem reduces to limiting the number of iterations required to attain the 1% tolerance limit. Further, here we make an assumption that *the relation between clock-to-output delay and setup clearance is monotonous*, i.e. it is always the case that the later the input arrives in the present clock cycle, the longer the clock-to-output delay will be, rather than exhibiting some local or global minima/maxima. This will allow the use of bisection facility of SPICE, namely a form of binary search, to be used.

Still, for a large cell library, the total number of iterations may be large. For this reason, although for linear model, setup time could, for instance, be modeled as dependent on input transition delay, the long simulation time needed largely renders this idea impractical. It is much more practical to simply use a reasonably pessimistic value, which may be obtained using a rather pessimistic assumption on clock rise/fall time.

The characterization work to have been performed used the following parameters in characterization.

– Evaluation range: -2ns to 2ns (it is assumed that setup time is in [-2ns, +2ns] range).
– Initial guess: -2ns.
– Number of iterations: ≤ 20. This should result in a precision of better than 1fs.
– Clock pulse 0%-to-100% rise/fall time: 0.5ns for 0.5μm technology, 0.4ns for 0.35μm technology, or in the same ballpark with clock-to-output delay.

Several words may need to be said on the choice of search range and initial guess. Returning to Figure 4.6, if the values of t_{su2} is made sufficiently large and t_{su3} is made sufficiently small (or even negative), we could be sure that $t_{su3} < t_{su} < t_{su2}$. If we arbitrarily choose, say, t_{su2} = 1ns and t_{su3} = 0ns, we could almost be assured that the aforementioned triple inequality will be satisfied, since most submicron flip-flops have subnanosecond setup times (hence, $t_{su} < t_{su2}$), and all master-slave structures have positive setup times (hence, $t_{su} > t_{su3}$). However, in general, we may need to be even more conservative in setting the borders of our range, since on one end some particularly complicated sequential elements, such as flip-flops with embedded combinational logic, may have setup times exceeding 1ns, while on the other end some novel flip-flops actually have positive setup times (i.e. even input transitions coming after clock transitions may still be responded upon correctly). In view of these facts, it is more prudent to use the following values:

t_{su3} = -2ns

t_{su2} = 2ns

Note that the setup times showed in Figures 4.6 and 4.7 are positive, hence not showing the negative setup times which we may encounter. Indeed, again, for our static devices, their setup times are always nonegative, but in general prudence may be warranted.

4. 2. 5. 2. SPICE Example

As an example, the following SPICE file performs setup time determination for simple 0.35μm D flip-flop wdp_2 – with comments added.

```
wdp_2 setup time characterization
.inc wcellsnet    ** the subckt netlists for 0.35um tech are stored here
```

```
.param lh='3.3v'
.param ll='0.0v'
vdd dd gnd lh
.param tsetupr=opt_tsetupr(-2ns, -2ns, 2ns)  ** initial guess is –2ns
.param tsetupf=opt_tsetupf(-2ns, -2ns, 2ns)

.op
.option post nomod accurate dcstep=1e-3 gmindc=1e-12 optlst=1 relv=1e-4
relvar=1e-2
** set the relative accuracy requirement to 0.1% (i.e. relv=1e-4)

** reference delay generator **
vdref dref gnd pwl (0ns ll 25ns ll 25.4ns lh 65ns lh 65.4ns ll)
xref dd dref ck qref qbref wdp_2
.measure vinit avg v(qref) from=4.5ns to=5.5ns
.measure prob_fall_adj param='(vinit-(lh/2))/(1v)'
.measure adj_sign param='sgn(prob_fall_adj)'
.measure fall_adjust param='max(adj_sign, 0)'
** the above parts determine whether the initial state of ff
** is high or low. If high, we must add 1 to number of fall
** transitions.
.measure tdr_ref trig v(ck) val='0.5*lh' rise=2 targ v(qref) val='0.5*lh'
+ rise=1
.measure tdf_ref trig v(ck) val='0.5*lh' rise=4 targ v(qref) val='0.5*lh'
+ fall='1+fall_adjust'
.tran 0.025ns 80ns

** setup measurement
vclk ck gnd pulse (ll lh 10ns 0.4ns 0.4ns 9.6ns 20ns)
vd d gnd pwl (0ns ll '30ns-tsetupr' ll '30.4ns-tsetupr' lh '70ns-tsetupf'
+ lh '70.4ns-tsetupf' ll)
xtested dd d ck qt qtb wdp_2

** Determination of setup time for rising input
.model=setuprmodel opt method=bisection itropt=20
** 20 iterations should be adequate – precision to several fs
.tran 0.025ns 90ns sweep optimize=opt_tsetupr result=sdr model=setuprmodel
.measure sdr trig v(ck) val='0.5*lh' rise=2 targ v(qt) val='0.5*lh'
+ rise=1 goal='1.01*tdr_ref'
** the goal is that sdr = ck-to-q delay within 1% of reference case
.measure src_setuprise_d param='tsetupr'
** if the aim is satisfied, just accept tsetupr as our estimate

** Then, do the same for fall transition.
.model=setupfmodel opt method=bisection itropt=20
.tran 0.025ns 110ns sweep optimize=opt_tsetupf result=sdf model=setupfmodel
.measure sdf trig v(ck) val='0.5*lh' rise=4 targ v(qt) val='0.5*lh'
+ fall='1+fall_adjust' goal='1.01*tdf_ref'
.measure src_setupfall_d param='tsetupf'

.end
```

The output file contains the following lines, among others (comments added):

```
******  transient analysis               tnom=  25.000 temp=  25.000
.
.
.
  tdr_ref=  2.3599E-10  targ=  3.0436E-08   trig=  3.0200E-08  ** reference
delays for output
  tdf_ref=  3.4499E-10  targ=  7.0545E-08   trig=  7.0200E-08  ** rising and
falling
.
.
.
entering lmopt
  parm names init guess, lower, upper bounds
tsetupr            -2.0000E-09   -2.0000E-09    2.0000E-09        0.
** bisection analysis for rising input setup time detremination
** initial guess is –2ns, as specified

bisec-opt.  iter =   1 xlo =  -2.00000E-09  xhi =   2.00000E-09
                         x =  -2.00000E-09 xnew =   2.00000E-09   ** new
guess is 2ns
                       err =   -83.900

bisec-opt.  iter =   2 xlo =  -2.00000E-09  xhi =   2.00000E-09
                         x =   2.00000E-09 xnew =        0.      ** next
guess is 0ns
                       err =   9.89874E-03
.
.
.
bisec-opt.  iter =  11 xlo =   3.82813E-10  xhi =   3.90625E-10
                         x =   3.82813E-10 xnew =   3.86719E-10
                       err =  -4.59668E-05

         optimization completed, the condition
         relin =  1.0000E-03 is satisfied

         optimization completed, the condition
         relout =  1.0000E-03 is satisfied

**** optimized parameters opt_tsetupr

.param tsetupr       = 382.8125p      ** required relative error attained
after only 11 iterations
.
.
.
******  transient analysis               tnom=  25.000 temp=  25.000
.
.
.
  tdr_ref=  2.3598E-10  targ=  3.0436E-08   trig=  3.0200E-08
  tdf_ref=  3.4499E-10  targ=  7.0545E-08   trig=  7.0200E-08
  sdr=  2.3835E-10  targ=  3.0438E-08   trig=  3.0200E-08
** compare tdr_ref and sdr: differ within 1%
  src_setuprise_d=  3.8281E-10
.
.
```

```
.
entering lmopt
  parm names init guess, lower, upper bounds
tsetupf            -2.0000E-09   -2.0000E-09    2.0000E-09        0.
** now for falling input setup time

bisec-opt.  iter =   1 xlo =  -2.00000E-09  xhi =   2.00000E-09
                         x =  -2.00000E-09 xnew =   2.00000E-09
                       err =   -57.389

bisec-opt.  iter =   2 xlo =  -2.00000E-09  xhi =   2.00000E-09
                         x =   2.00000E-09 xnew =        0.
                       err =   9.85833E-03
.
.
.
bisec-opt.  iter =  18 xlo =   4.37561E-10  xhi =   4.37622E-10
                         x =   4.37561E-10 xnew =   4.37592E-10
                       err =  -3.26674E-06
** this time needs 18 iterations

         optimization completed, the condition
         relin =  1.0000E-03 is satisfied

         optimization completed, the condition
         relout =  1.0000E-03 is satisfied


**** optimized parameters opt_tsetupf

.param tsetupf       = 437.5610p
.
.
.
Opening plot unit= 15
file=./wdp_2_setup.tr2

******
wdp_2 setup time characterization - part 1
 ******  transient analysis               tnom=  25.000 temp=  25.000
******
.
.
.
  tdf_ref=  3.4491E-10  targ=  7.0545E-08   trig=  7.0200E-08
  sdf=  3.4836E-10  targ=  7.0548E-08   trig=  7.0200E-08
** sdf=348.36ps, compare with the 344.91ps reference delay
  src_setupfall_d=  4.3756E-10
.
.
```

In some occasions, the desired value of relative error may still not be attained after 20 iterations. However, even if that is the case, the resulting estimate should still be accurate to several femtoseconds, and the warning messages could simply be disregarded.

4. 2. 6. Tristate Cell Timing and Input/Output Capacitance Characterization

Characterization of tristate cells is basically the same as characterization of a standard combinational cell, except for three new considerations:

1. Enable- and disable-times must also be considered.
2. Because of possibilities of charge injection, a tristate buffer cannot be meaningfully characterized by simulating it in isolation; rather, at least two cells must be tested in parallel, driving the same node.
3. Since typically several tristate buffers are ganged, i.e. several of them drive the same node (hence when they are not driving the node, they present a load to other cells driving the node), their **output capacitance** must also be determined.

Following the first point, several simulation tools add several definitions for the delay types encountered, as follows:

Transition type	**Name**
Enable/disable signal to Z-to-0 at output	enable_rise delay
Enable/disable signal to Z-to-1 at output	enable_fall delay
Enable/disable signal to 0-to-Z at output	disable_rise delay
Enable/disable signal to 1-to-Z at output	disable_fall delay

Note that several synthesis tools may not define disable times, under the assumption that disable times are always longer than enable times, hence they will not be observed nor have any effects during normal circuit operation.

The second point may merit more clarification. Let us consider the following tristate buffer:

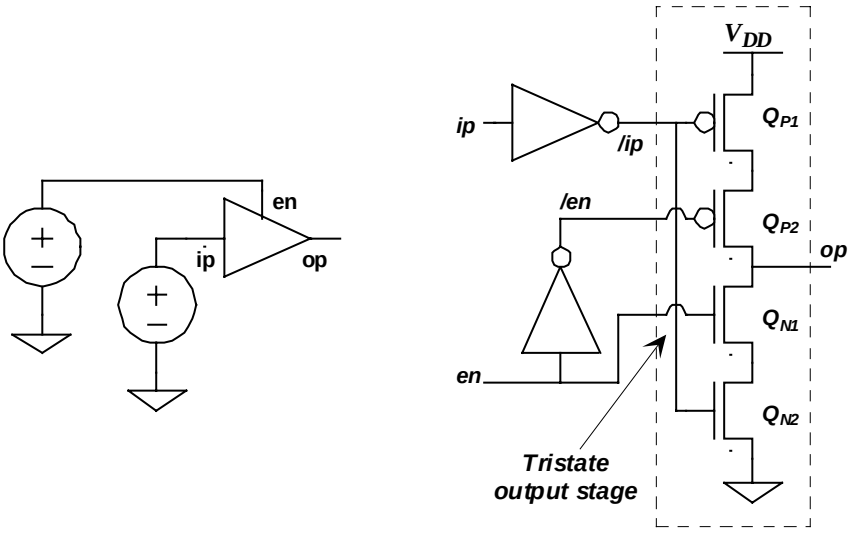


Figure 4. 17. Noninverting tristate buffer. Left: circuit connection. Right: Internal circuit of the buffer cell.

The waveform used is as follows:

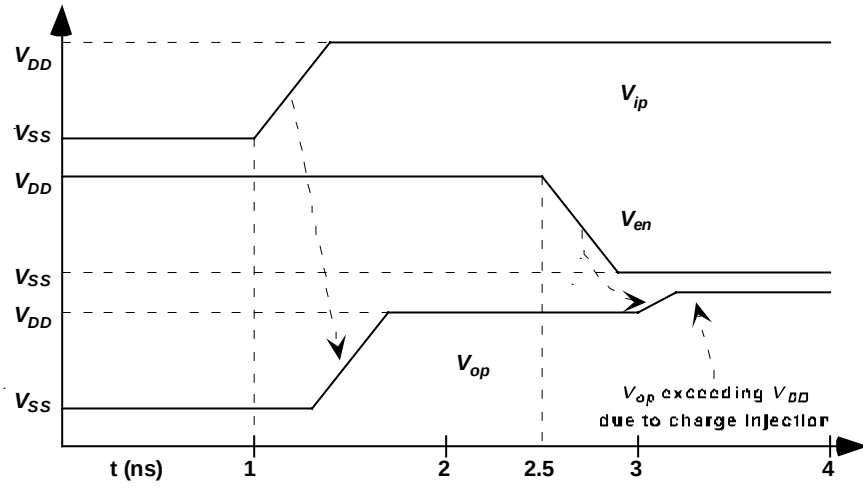


Figure 4.18. Waveforms used. The disabling transition at 2.5ns causes V_{op} to exceed the rail.

As shown above, while the ***ip*** input transition at 1ns does not cause anything unusual (it simply causes a normal transition at output), the disabling transition (***en*** deasserted) at 2.5ns causes the output ***op*** to exceed the V_{DD} rail. What happens is, when ***en*** is deasserted, ***op*** is left floating (electrically cut off from other nodes). However, just before that occurs, charge accumulating in the channels of Q_{P2} and Q_{N1} will be partly dumped into ***op***. Although the charge

from p- and n-devices are opposite, they do not cancel each other in this case, since the sizes of the p- and n-devices differ significantly, hence their channel capacitances, and therefore also the accumulated charge, also differ in magnitude. When ***op*** gets cut off, the charge will be left there with nowhere to go, hence altering V_{op} – in this case, causing it to rise beyond the rail. In other words, the voltage ***op*** will not be strictly a logic value, and cannot be used to define a logic propagation delay. To be used in logic delay calculations, signals must continue to maintain valid logic levels when not in transition, i.e. stay within the rails. This is guaranteed by connecting the outputs of several tristate cells to the same node as in actual use. Intrinsic delays are then determined through interpolation, by comparing the delays of two different circuits as follows:

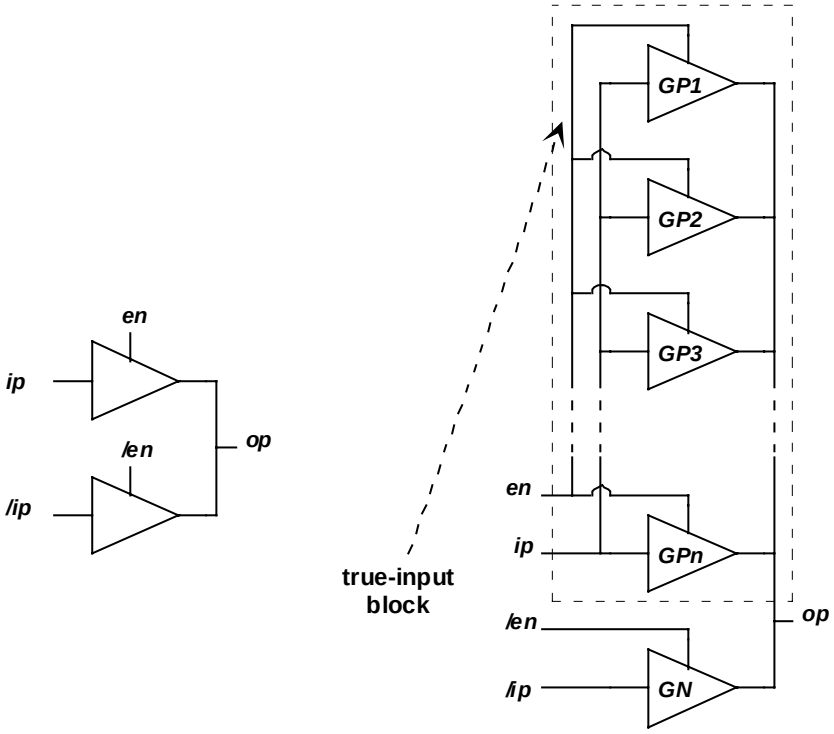


Figure 4. 19. Circuit for delay determination. Left: 1-vs-1 circuit. Right: n-vs-1 circuit.

4. 2. 6. 1. Intrinsic Delays and Enable Delay Determination

The above two circuits will exhibit different delays. Let us focus to the delay parameters of the true-input blocks. The **enable delays** of the n-vs-1 circuit will be shorter than those of the 1-vs-1 circuit, since for the n-vs-1 circuit the charge dumped from (or into) the output capacitances of *GN* when *GN* is disabled will be absorbed (or provided) by n buffer cells (*GP1*

through *GPn*), while for the 1-vs-1 circuit the charge will be absorbed (or provided) by just one buffer. Therefore, the additional delay (in addition to intrinsic delays) exhibited by the n-vs-1 circuit will only be (*1/n*) times as large as exhibited by the 1-vs-1 circuit. Therefore, assuming that the input waveforms are proper for **intrinsic delay determination**, it could be shown that the following two relations are satisfied:

$$delay_{intrinsic} = delay_{left} - \Delta_{delay} \ . \tag{4.17}$$

$$\Delta_{delay} = (delay_{left} - delay_{right}) . \ n / (n-1) \ . \tag{4.18}$$

Since $delay_{right}$ and $delay_{left}$ could be measured, while *n* is known, intrinsic enable delays could be found. Additionally, ***ip→op*** delays could also be found using the same formulae, again by observing the delay of the true-input block. Note that if high accuracy is desired, the above procedure could be repeated with several different values of *n*, and the resulting intrinsic delays are compared and, if necessary, averaged.

To determine output resistance and slope sensitivities of enable delays (for ***ip→op*** delays will be determined in a more conventional fashion), the same scheme is used.

4. 2. 6. 2. Determination of ip→op Output Resistances

For **output resistance** determination, using the identical waveforms as used for intrinsic delays determination, we use the ***1-vs-1*** circuit, with a C_{load} connected to the output.

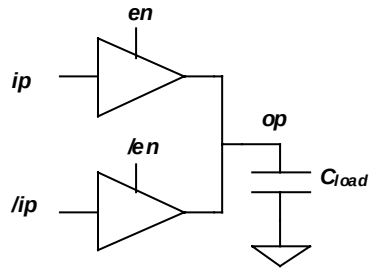


Figure 4. 20. Circuit used for output resistance determination

The enable delays will extend by a transition delay (= *delay with load capacitor – delay without load capacitor*). Hence, we simply have:

$$R_{out} = (delay_{with\_Cload} - delay_{without\_Cload}) / C_{load} \quad (4.19)$$

4. 2. 6. 3. Input Capacitance Determination

The determination of input capacitances of tristate buffers uses the 1-vs-1 circuit of Figure 3. left. The measurement process and the waveforms used should satisfy the following conditions:

1. Measurement should be performed only for inputs which cause output changes. Since all inputs are complementary in our case, this is assured if C_{ip} is measured only when the buffer is enabled..
2. For each input pin, measurement should be performed both for input transitions which cause rising output transition and for input transitions which cause falling output transition, and the result should be averaged. For enable pins, we have four possibilities: disable→rise, disable→fall, enable→rise, and enable→fall.

4. 2. 6. 4. Output Capacitance Determination

The determination of output capacitance is performed in a similar manner with input capacitance determination, namely by applying a voltage waveform to the output pin and measuring the amount of charge injected. Both rising- and falling-waveform should be used. The tristate cell examined must be in disabled state to avoid direct path to V_{DD} / ground, which will cause indefinite capacitance. Also, measurement should be performed for both ***ip***=1 and ***ip***=0 case. One circuit configuration which may be used is as follows:

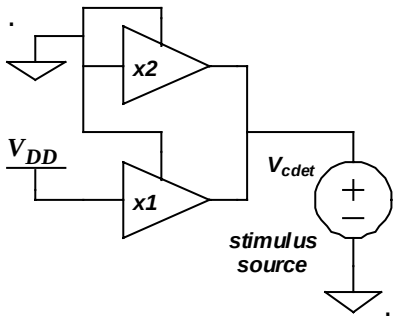


Figure 4. 21. Circuit for C_{op} measurement (high-active enablers)

The following type of waveform may be used for V_{cdet}:

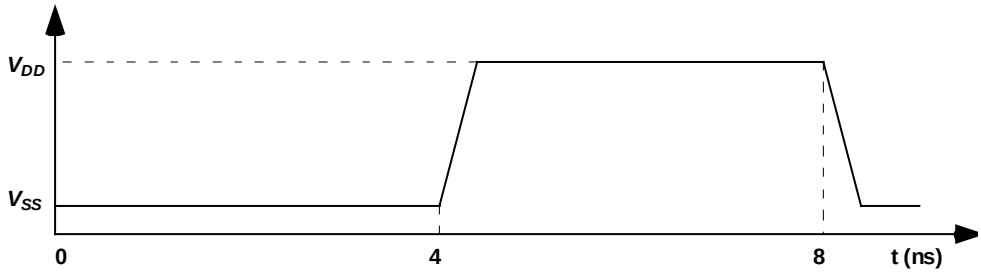


Figure 4. 22. V_{cdet} waveform for C_{op} determination

In this scheme, ***x1*** is instantiated to measure charge accumulated (or dumped) when the input=H, while ***x2*** is for the input=L case. Further, we may measure the magnitude of the charge injected into ***x1*** and ***x2*** (or sourced from them) for both rising-and falling stimuli (at 4ns and 8ns, respectively), and we already obtain the sum of the charge accumulations for all four combinations of input values and stimulus transitions; to obtain the estimate for C_{op}, this is simply averaged (i.e. divided by four) and then divided by the magnitude of the stimulus. The SPICE code may look as follows (taking measurement period as 4ns):

```
.measure qrise integral i(vcdet) from=4ns to=8ns
.measure qfall integral i(vcdet) from=8ns to=12ns
.measure qav param='0.25*(abs(qrise)+abs(qfall))'
.measure cop param='qav/vdd'
```

4. 2. 6. 5. Measurement of Output Resistances for Enable Delays

For enable/disable delays, output resistances could also be performed by using the circuit at Figure 4 as is the case for *ip→op* delays. However, here a different method is presented. Here instead of performing new measurements, the results from intrinsic delay measurement process are used. Specifically, referring to eqs. (1) – (2), Δ_{delay} could be interpreted as the transition delay at output op, exhibited by the enable delay of a buffer, due to the presence of C_{out} of the other buffer. Hence:

$$R_{out,enable} = \Delta_{delay,enable} / C_{out} \qquad (4.20)$$

4. 2. 6. 6. Measurement of Disable Delays

This part is not performed, as the disabling process is not directly observable. It is assumed that the disabling process is sufficiently fast to avoid fighting.

4. 2. 6. 7. SPICE Example

As an example, the following SPICE file performs the timing characterization of the noninverting tristate buffer hbfzp. Figure 4.23 shows the circuit for slope delay determination. This circuit will later also be used for power charactreization. Note that only the 1-vs-1 pair is shown here. Actually, 2-vs-1 circuits were also used to allow the use of formulae (4.17)-(4.20).

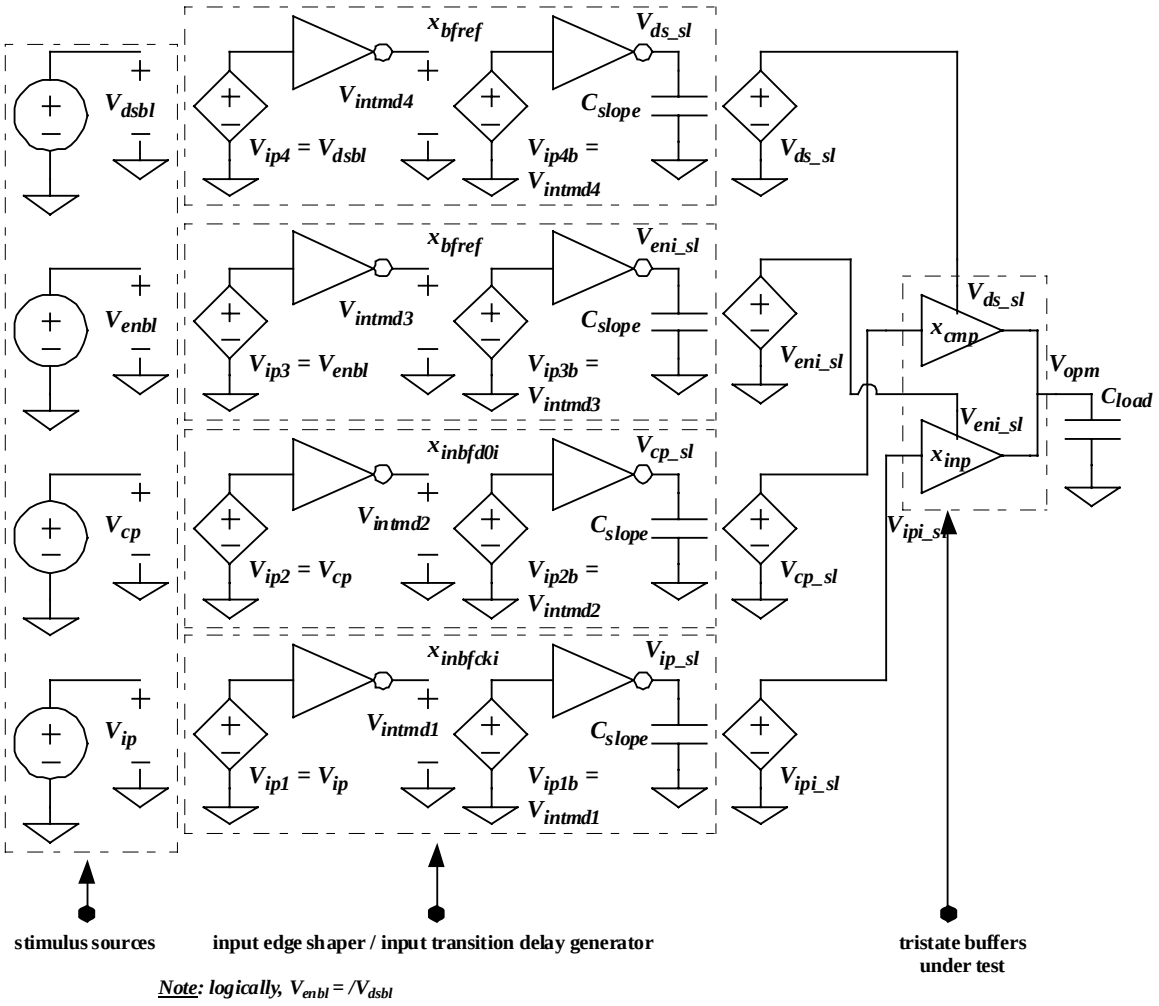


Figure 4.23. Circuit for Slope Delay Determination of Tristate Buffers, shown here with 1-vs-1 Circuit

The SPICE file is as follows:

```
.inc hiv.l
.inc hbfzp.l

** Power Supply
.param spl='3.3v'
.param ss='0v'
.param loaderc='75fF'
.param sloperc='75fF'
.param powerc='100fF'
vdd dd gnd spl

** subcircuits
```

```
.subckt nlhiv dd ip op
eip ipi gnd ip gnd 1
xiv dd ipi op hiv
.ends nlhiv
.subckt nlbuf dd ip op
x1 dd ip ipi nlhiv
x2 dd ipi op nlhiv
.ends nlbuf
.subckt nlhbfzp dd ip en op
eip ipi gnd ip gnd 1
een eni gnd en gnd 1
xbz dd ipi eni op hbfzp
.ends nlhbfzp
** stimuli
vip ip gnd pwl (0ns ss 12ns ss 12.2ns spl 24ns spl 24.1ns ss 32ns ss 32.2ns
spl
+ 44ns spl 44.1ns ss 68ns ss 68.2ns spl 84ns spl 84.1ns ss 88ns ss 88.2ns spl
+ 96ns spl 96.1ns ss 100ns ss 100.2ns spl)
** ci for delay determination
vci ci gnd pwl (0ns spl 12ns spl 12.1ns ss 24ns ss 24.2ns spl 32ns spl 32.1ns
ss
+ 44ns ss 44.1ns spl 48ns spl 48.2ns ss 68ns ss 68.1ns spl 92ns spl 92.2ns
ss)
** ci for power characterization
vcp cp gnd pwl (0ns spl 12ns spl 12.1ns ss 24ns ss 24.2ns spl 32ns spl 32.1ns
ss
+ 44ns ss 44.2ns spl 48ns spl 48.1ns ss 68ns ss 68.2ns spl 92ns spl 92.1ns
ss)
.param enbl='spl'
.param dsbl='ss'
** enabler/disabler
venbl en gnd pwl (0ns dsbl 4ns dsbl 4.2ns enbl 8ns enbl 8.1ns dsbl 16ns dsbl
+ 16.2ns enbl 20ns enbl 20.1ns dsbl 28ns dsbl 28.2ns enbl 36ns enbl 36.1ns
dsbl
+ 40ns dsbl 40.2ns enbl 52ns enbl 52.1ns dsbl 56ns dsbl 56.2ns enbl 60ns enbl
+ 60.1ns dsbl 64ns dsbl 64.2ns enbl 72ns enbl 72.1ns dsbl 76ns dsbl 76.2ns
enbl
+ 80ns enbl 80.1ns dsbl)
vdsbl ds gnd pwl (0ns enbl 4ns enbl 4.1ns dsbl 8ns dsbl 8.2ns enbl 16ns enbl
+ 16.1ns dsbl 20ns dsbl 20.2ns enbl 28ns enbl 28.1ns dsbl 36ns dsbl 36.2ns
enbl
+ 40ns enbl 40.1ns dsbl 52ns dsbl 52.2ns enbl 56ns enbl 56.1ns dsbl 60ns dsbl
+ 60.2ns enbl 64ns enbl 64.1ns dsbl 72ns dsbl 72.2ns enbl 76ns enbl 76.1ns
dsbl
+ 80ns dsbl 80.2ns enbl)
** edge-degraded version of input stimuli
** for intrinsic delays
xip dd ip ipi nlbuf
xen dd en eni nlbuf
xipn dd ci cii nlbuf
xenn dd ds dsi nlbuf
** for slope delays
xips dd ip ipi_sl nlbuf
cips ipi_sl gnd sloperc
xens dd en eni_sl nlbuf
cens eni_sl gnd sloperc
xcis dd ci ci_sl nlbuf
```

```
ccis ci_sl gnd sloperc
xcps dd cp cp_sl nlbuf
ccps cp_sl gnd sloperc
xdss dd dsi ds_sl nlbuf
cdss ds_sl gnd sloperc

** Cin and Cout determination **
** Cin determination **
eipcd ipcd gnd ip gnd 1
eenblcd encd gnd en gnd 1
ecipcd cipcd dd ipcd gnd -1
ecenblcd cencd dd encd gnd -1
xincd dd ipcd encd opcdet hbfzp
xcmcd dd cipcd cencd opcdet hbfzp
.measure qipr integral i(eipcd) from=32ns to=36ns
.measure qipf integral i(eipcd) from=44ns to=48ns
.measure qenr integral i(eenblcd) from=40ns to=44ns
.measure qenf integral i(eenblcd) from=4ns to=8ns
.measure qdsr integral i(eenblcd) from=8ns to=12ns
.measure qdsf integral i(eenblcd) from=36ns to=40ns
.measure qip param='(abs(qipr)+abs(qipf))/2'
.measure qen param='(abs(qenr)+abs(qenf)+abs(qdsr)+abs(qdsf))/4'
.measure cip param='qip/spl'
.measure cen param='qen/spl'
** Cout determination **
vopcd opcd gnd pwl (0ns ss 4ns ss 4.1ns spl 8ns spl 8.2ns ss)
xopcdet1 dd dd gnd opcd hbfzp
xopcdet2 dd gnd gnd opcd hbfzp
.measure qopr integral i(vopcd) from=4ns to=8ns
.measure qopf integral i((vopcd) from=8ns to=12ns
.measure qop param='(abs(qopr)+abs(qopf))/4'
.measure cout param='qop/spl'

** intrinsic delay determination - first step **

** 1-vs-1 circuit
xin dd ipi eni op nlhbfzp
xcm dd ci dsi op nlhbfzp
.measure tenr11 trig v(eni) val='0.5*spl' rise=2 targ v(op) val='0.5*spl'
rise=2
.measure tenf11 trig v(eni) val='0.5*spl' rise=1 targ v(op) val='0.5*spl'
fall=1

** 2-vs-1 circuit
xi2a dd ipi eni op2 nlhbfzp
xi2b dd ipi eni op2 nlhbfzp
xc2 dd ci dsi op2 nlhbfzp
.measure tenr21 trig v(eni) val='0.5*spl' rise=2 targ v(op2) val='0.5*spl'
+ rise=2
.measure tenf21 trig v(eni) val='0.5*spl' rise=1 targ v(op2) val='0.5*spl'
+ fall=1
.measure dtenr param='(tenr11-tenr21)*2'
.measure dtenf param='(tenf11-tenf21)*2'

** estimates for enable / disable intrinsic delays **
.measure tenr_int param='tenr11-dtenr'
.measure tenf_int param='tenf11-dtenf'
```

```
** ip -> op delay characterization
** intrinsic delays **
xi_int dd ipi dd opi_int nlhbfzp
.measure tripop_int trig v(ipi) val='0.5*spl' rise=1 targ v(opi_int)
+ val='0.5*spl' rise=1
.measure tfipop_int trig v(ipi) val='0.5*spl' fall=1 targ v(opi_int)
+ val='0.5*spl' fall=1
** output resistances **
xi_res dd ipi dd opi_res nlhbfzp
ci_res opi_res gnd loaderc
.measure tripop_res trig v(ipi) val='0.5*spl' rise=1 targ v(opi_res)
+ val='0.5*spl' rise=1
.measure tfipop_res trig v(ipi) val='0.5*spl' fall=1 targ v(opi_res)
+ val='0.5*spl' fall=1
.measure resrise_ipop param='(tripop_res-tripop_int)/loaderc'
.measure resfall_ipop param='(tfipop_res-tfipop_int)/loaderc'
** slope sensitivity **
xi_slp dd ipi_sl dd opi_slp nlhbfzp
.measure tr_bi trig v(xip.ipi) val='0.5*spl' fall=1 targ v(ipi) val='0.5*spl'
+ rise=1
.measure tf_bi trig v(xip.ipi) val='0.5*spl' rise=1 targ v(ipi) val='0.5*spl'
+ fall=1
.measure tr_bs trig v(xips.ipi) val='0.5*spl' fall=1 targ v(ipi_sl)
+ val='0.5*spl' rise=1
.measure tf_bs trig v(xips.ipi) val='0.5*spl' rise=1 targ v(ipi_sl)
+ val='0.5*spl' fall=1
.measure dtr param='tr_bs-tr_bi'
.measure dtf param='tf_bs-tf_bi'
.measure tripop_slp trig v(ipi_sl) val='0.5*spl' rise=1 targ v(opi_slp)
+ val='0.5*spl' rise=1
.measure tfipop_slp trig v(ipi_sl) val='0.5*spl' fall=1 targ v(opi_slp)
+ val='0.5*spl' fall=1
.measure dsr param='tripop_slp-tripop_int'
.measure dsf param='tfipop_slp-tfipop_int'
.measure sloperise_ipop param='dsr/dtr'
.measure slopefall_ipop param='dsf/dtf'

** enable output resistances **
.measure resrise_en param='dtenr/cout'
.measure resfall_en param='dtenf/cout'

** last part - finding slope sensitivities for enable/disable **
** using modified copy of intrinsic delay determination - first step **

xins dd ipi_sl eni_sl op_sl nlhbfzp
xcms dd ci_sl ds_sl op_sl nlhbfzp

.measure tenr11_sl trig v(eni_sl) val='0.5*spl' rise=2 targ v(op_sl)
+ val='0.5*spl' rise=2
.measure tenf11_sl trig v(eni_sl) val='0.5*spl' rise=1 targ v(op_sl)
+ val='0.5*spl' fall=1
.measure tdsr11_sl trig v(eni_sl) val='0.5*spl' fall=1 targ v(op_sl)
+ val='0.5*spl' rise=1
.measure tdsf11_sl trig v(eni_sl) val='0.5*spl' fall=2 targ v(op_sl)
+ val='0.5*spl' fall=3
.measure sloperise_enr param='(tenr11_sl-tenr11)/dtr'
```

```
.measure slopefall_enf param='(tenf11_sl-tenf11)/dtr'

.option post nomod accurate
.option dcstep=1e-3 gmindc=1e-12
.tran 0.025ns 104ns

.end
```

The output contains the following lines:

```
******  transient analysis               tnom=  25.000 temp=  25.000
******
  qipr= -1.3818E-13  from=  3.2000E-08     to=  3.6000E-08
  qipf=  1.3846E-13  from=  4.4000E-08     to=  4.8000E-08
  qenr= -1.6149E-13  from=  4.0000E-08     to=  4.4000E-08
  qenf= -2.0345E-13  from=  4.0000E-09     to=  8.0000E-09
  qdsr=  2.0370E-13  from=  8.0000E-09     to=  1.2000E-08
  qdsf=  1.6279E-13  from=  3.6000E-08     to=  4.0000E-08
  qip=  1.3832E-13
  qen=  1.8286E-13
  cip=  4.1916E-14    ** Estimated input capacitance
  cen=  5.5411E-14
  qopr= -1.9534E-13  from=  4.0000E-09     to=  8.0000E-09
  qopf=  1.9582E-13  from=  8.0000E-09     to=  1.2000E-08
  qop=  9.7790E-14
  cout=  2.9633E-14    ** Estimated output capacitance
  tenr11=  2.2641E-10  targ=  1.6429E-08   trig=  1.6202E-08
  tenf11=  9.6037E-11  targ=  4.2983E-09   trig=  4.2023E-09
  tenr21=  1.9367E-10  targ=  1.6396E-08   trig=  1.6202E-08
  tenf21=  6.9571E-11  targ=  4.2718E-09   trig=  4.2023E-09
  dtenr=  6.5491E-11
  dtenf=  5.2931E-11
  tenr_int=  1.6092E-10    ** Intrinsic enable_rise time
  tenf_int=  4.3106E-11    ** Intrinsic enable_fall time
  tripop_int=  2.7025E-10  targ=  1.2473E-08   trig=  1.2202E-08
  tfipop_int=  1.9648E-10  targ=  2.4351E-08   trig=  2.4154E-08
  tripop_res=  4.1742E-10  targ=  1.2620E-08   trig=  1.2202E-08
  tfipop_res=  2.7453E-10  targ=  2.4429E-08   trig=  2.4154E-08
  resrise_ipop=  1.9623E+03  ** rise_resistance of op from input ip
  resfall_ipop=  1.0406E+03  ** fall_resistance of op from input ip
  tr_bi=  5.7224E-11  targ=  1.2202E-08   trig=  1.2145E-08
  tf_bi=  4.2196E-11  targ=  2.4154E-08   trig=  2.4112E-08
  tr_bs=  1.3479E-10  targ=  1.2280E-08   trig=  1.2145E-08
  tf_bs=  8.9407E-11  targ=  2.4202E-08   trig=  2.4112E-08
  dtr=  7.7565E-11
  dtf=  4.7211E-11
  tripop_slp=  2.8641E-10  targ=  1.2566E-08   trig=  1.2280E-08
  tfipop_slp=  2.1550E-10  targ=  2.4417E-08   trig=  2.4202E-08
  dsr=  1.6161E-11
  dsf=  1.9020E-11
  sloperise_ipop=  2.0835E-01    ** slope_rise for input -> output
  slopefall_ipop=  4.0287E-01    ** slope_fall for input -> output
```

```
resrise_en=  2.2101E+03  ** rise_resistance for enable_rise
resfall_en=  1.7862E+03  ** fall_resistance for enable_fall
tenr11_sl=  2.4808E-10  targ=  1.6528E-08   trig=  1.6280E-08
tenf11_sl=  1.4369E-10  targ=  4.4236E-09   trig=  4.2799E-09
tdsr11_sl=  3.6022E-10  targ=  8.5617E-09   trig=  8.2015E-09
tdsf11_sl=  2.7778E-10  targ=  2.0479E-08   trig=  2.0202E-08
sloperise_enr=  2.7927E-01  ** slope_rise for enable -> rise
slopefall_enf=  6.1440E-01  ** slope_fall for enable -> fall

       ***** job concluded
```

Chapter V

Power Characterization

In this chapter, we will discuss the power characterization of a library cell. Here we will limit our discussion to static CMOS logic circuit.

5. 1. Basics of Power Dissipation

The power dissipation of a cell is divided into two parts – static and dynamic power.

1. Static power: This is the power which is dissipated when no switching activity is occurring. For static complementary CMOS cells, this is mostly due to leakage power and subthreshold conduction current.
2. Dynamic power: This is dissipated when switching activities are occurring. For static CMOS circuits, dynamic power usually comprises a large majority of the overall power dissipation. This mode of power can be further divided into two categories:
   a) *Switching Power:* This is the energy / power dissipated by a cell due to the charging and discharging of external capacitive loads being driven by the cell. This is already defined by the size of input capacitances of the cells as determined in capacitance/timing characterization as described in the previous chapter. The same capacitance value is used here.
   b) *Internal Power:* This is the part of dynamic energy which is dissipated by the cell in addition of switching power. Physically, internal energy is dissipated, in pulses, due to short-circuit conditions that occur during switching and due to charging/discharging of the internal capacitances of the cell itself.

To better explain the above concepts, let us use a specific example. Let us use the following 2-input complementary static CMOS NAND gate as an example. To simplify discussion, let us assume that the voltage at the input pins eventually settle at the rail voltages (0V or V_{DD}).

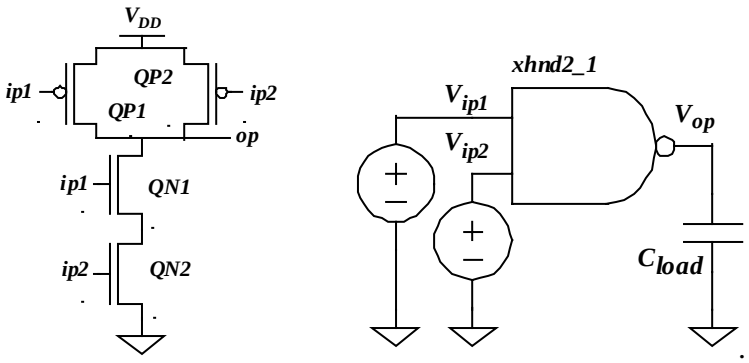


Figure 5.1: a 2-input complementary static CMOS NAND gate. **Left:** transistor level schematic. **Right:** Schematic of a the cell in use and its surrounding. C_{load} represent the capacitive load posed by another cell being driven by the NAND gate.

5. 1. 1. Static Power

First, let us analyze the power dissipation of the NAND gate if the voltages at the input pins ip1 and ip2 do not make any logic transitions for a long time. In that case, although some of the transistors may be turned on, there should not be any DC path from V_{DD} to ground, and thus no power dissipation would occur. However, actually some leakage current will flow and cause power dissipation. This is the source of **static power dissipation**, which could therefore be equated with leakage power for our complementary CMOS case. Note that this static power dissipation may vary depending on the voltages in the input pins.

Note that we will have four different cases of static power dissipation, namely the cases when logical values at {ip1, ip2} = (0, 0), (0, 1), (1, 0), and (1,1), respectively. Let us call them P_{s00}, P_{s01}, P_{s10}, and P_{s11} respectively. Then, assuming that all those four input combinations are equally likely, we have:

average static power, (op = high) = $(P_{s00} + P_{s01} + P_{s10}) / 3$ (5.1)

average static power, (op = low) = P_{s11} (5.2)

average static power =

$[(3\times$average static power, (op = high))+average static power, (op = low)$] / 4$ (5.3)

5. 1. 2. Dynamic Power

When the input voltages change, then the output voltage V_{op} – to a first order approximation – may or may not change (actually, a more detailed analysis may show that any input change will automatically cause output change, although the change may not qualify as a logic transition). Let us analyze those two cases separately. Let us assume that only one input transition could occur at any given time (i.e. no two inputs will experience logic transition at the same time), and that no new transition will occur at input before output transitions caused by the previous input has completed.

First case. Logic transitions at input cause output logic transition. In this case, at some moment there would be short-circuit condition across the cell. Additionally, capacitances present at the cell's output (as well as other capacitances) may charge or discharge. This will cause power to be dissipated.

Second case. Logic transitions at input cause no output logic transition. As long as the input transitions do not cause any short-circuit or capacitance charging / discharging condition, such input transitions will not cause any power dissipation to take place. Actually, even such input transitions may cause some glitch at the output and therefore cause some power dissipation, although unless the cell has an internal load, this should be relatively insignificant compared with the first case.

Let us use, as a concrete example, a load capacitance of 50fF, and waveforms as in Figure 5. 2.

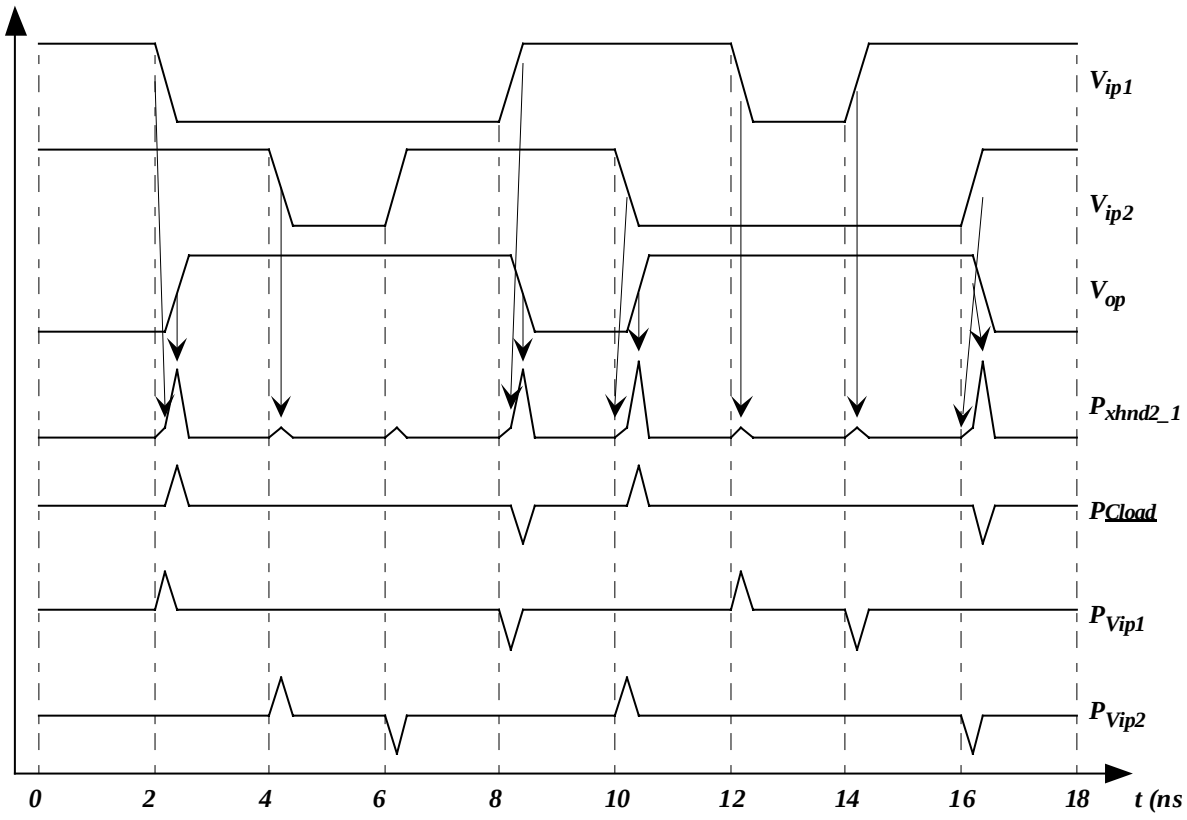


Figure 5. 2. Waveforms used in dynamic power measurement of a 2-input NAND gate.

For any given interval, we have:

Total Power = Static Power + Dynamic Power (5.4)

Dynamic Power = Switching Power + Internal Power (5.5)

Note that the model defined by the above two equations do not fully correspond to physical reality, as will be discussed later.

Static power. Note that we expect P_{xhnd2_1} = (static power) whenever switching is not taking place.

Power supplied by stimulus source. P_{Vip1} and P_{Vip2}, respectively, represent the amount of power received by the stimulus sources. Hence, the power supplied by those stimulus source

are $-P_{Vip1}$ and $-P_{Vip2}$, respectively. Note that in our case, those quantities could be positive or negative and over infinite time interval should ideally be zero (ignoring gate resistances, or more accurately conductances, of the transistors), since those stimulus sources are not connected to any energy dissipation element. In any case, for SPICE simulation purposes, those quantities are immaterial, as will be discussed later.

Total Power and Switching Power. Here P_{xhnd2_1} is the total resistive power dissipation, or let us simply call it power dissipation, occurring within the borders of the xhnd2_1 NAND gate, while P_{Cload} is the power supplied to the load capacitor C_{load}. The hills and peaks in P_{xhnd2_1} represent the dynamic energy (since they occur only in association with some logic transition in some signal), while the low plains represent the static energy.

Again, ***since the ideal capacitor is an energy storage element and not an energy dissipation element, P_{Cload} could be positive or negative, and over infinite period should average to zero; however, in relation to equation 5, this quantity is to be taken as positive always, i.e. we always use the absolute value of P_{Cload}***. The rationale is as follows: the capacitor itself cannot dissipate any energy, since it is simply an energy storage element. Further, our depiction of the capacitor as being connected only to op and ground is also not correct of real gates (instead of our stipulated capacitor), since the PMOS transistors in the gate also have capacitances, which should instead be connected to V_{DD}; however, we simply model those capacitances as a lumped capacitor connected to ground. Further, and most importantly, although the capacitor itself cannot dissipate any energy, the charge $\boldsymbol{Q = CV_{DD}}$ injected into, or leaving, the capacitor must still contains a positive quantity of energy $\boldsymbol{E = \frac{1}{2}QV_{DD} = \frac{1}{2}CV_{DD}^2}$, and this energy must come from somewhere (here it is from the power supply), and must be dissipated somewhere. In our idealized model, the only place this dissipation could occur is in the transistors (and interconnects) which comprise the gate, as they have resistance. Thus, the switching energy, and hence switching power, is not energy (or power) dissipated by the capacitor itself; rather, ***the switching power is a positive amount of power which, due to the presence of the capacitor, is dissipated by the resistive elements inside the logic cells connected to the capacitor***. Let us further note that, for the purpose of this measurement for gate-level modeling application, ***switching energy is said to be dissipated only if a full-swing rail-to rail***

transition occurs at the output; if such an output transition does not occur, all the energy dissipated should be lumped to internal energy. Or, in other words, switching energy is said to be dissipated only if logically useful transitions and / or hazards occur, and not when only glitches occur, where a hazard is defined as a full-swing but logically useless output transition, while a glitch is defined as a logically useless transition which does not swing fully rail-to-rail.

Further, when circuit simulation programs (such as SPICE) are used for measuring power associated with an element such as a MOS transistor, which is capable of resistively dissipating energy (rather than purely stores and releases energy), only resistive energy dissipation will be reported. Therefore, energy supplied to the cell by the stimulus sources V_{ip1} and V_{ip2} will not be included. If, instead of coming from independent voltage sources, those stimuli had instead come from other logic cells (which are themselves capable of resistive power dissipation), than the absolute values of P_{Vip1} and P_{Vip2} would have been the "switching energy" part of the power dissipation of those logic cells, rather than being part of our NAND gate xhnd2_1's power dissipation. For this reason, as stated before, as far as determining the power dissipation of xhnd2_1 is concerned with, P_{Vip1} and P_{Vip2} should simply be ignored.

Therefore, for any given period, we have:

P_{xhnd2_1} = total energy dissipation of xhnd2_1
= average leakage power + average $|P_{Cload}|$ + average internal power (5. 6)

Or, if we simply concentrate on the vicinity of an output transition and the associated input transition (where the static energy could be neglected), and measuring energy instead of power:

$$E_{xhnd2\_1} = \text{internal energy} + |P_{Cload}| \quad (5.7)$$

In other words, actually in our model, internal energy (power) represents not only short-circuit power, but also all dynamic energy not already accounted for by the switching energy.

Let us evaluate specific cases of dynamic energy dissipations, again as shown in **Figure 5. 2**.

1. at t = 12 ns, a falling transition occurs at ip1. This does not cause a full logic transition at the output (although actually some small glitch may occur). As we see in the curve (which is not according to scale), this input transition causes a small increase in P_{xhnd2_1}. Hence, if we take a time interval just from the start of input transition to just the end of the input and output transitions (say, from 2 ns to 3 ns) and integrate P_{xhnd2_1} w.r.t. time to obtain energies, the quantity $E_{hnd2,ip1f}$ obtained will be ***the internal energy of ip1 caused by falling ip1***. Let us use "hnd2" instead of "xhnd2_1" in the subscript for energy here, since the same quantity should be obtained for any identical hnd2 under identical situation. Note that the output transitions (glitches) are not visible in the curve for V_{op} in our graph, and we need to evaluate the SPICE waveform to find them.

2. Similarly, we obtained the internal energy of ip1 caused by rising ip1 $E_{hnd2,ip1r}$, as well as internal energy of ip2 caused by falling and rising ip2, $E_{hnd2,ip2f}$ and $E_{hnd2,ip2r}$, through the same procedures on the input transitions at 14 ns, 4 ns, and 6 ns, respectively.

3. Note that those four quantities found in **1** and **2** above are negligible in this case, and could well have been approximated as 0 with no significant loss of modeling accuracy. This is true for all basic combinational cells (i.e. ones which cannot be further subdivided into even smaller cells). However, this is not generally true for sequential cells and complex combinational cells, as will be apparent later.

4. at t = 2 ns, an input transition occurs at ip1, which causes rising transition at op. Therefore, if we take a time interval just from the start of input transition to just the end of the input and output transitions (say, from 2 ns to 3 ns) and integrate the power dissipations with respect to time to obtain energies, the quantity

$$E_{hnd2,r,ip1} = E_{hnd2} - | E_{Cload} | - E_{hnd2,ip1f} \quad (5.8)$$

is the ***rising transition energy of op caused by (falling) ip1***.

5. Similarly, we may find the other internal energies as follows:
   falling transition energy of op caused by (rising) ip1 caused by rising transition of ip1 at 8 ns
   $E_{hnd2,f,ip1} = E_{hnd2} - | E_{Cload} | - E_{hnd2,ip1r}$ (5.9)

   rising transition energy of op caused by (rising) ip1 caused by falling transition of ip2 at 10 ns
   $E_{hnd2,r,ip2} = E_{hnd2} - | E_{Cload} | - E_{hnd2,ip2f}$ (5.10)

   falling transition energy of op caused by (rising) ip1 caused by rising transition of ip2 at 16 ns
   $E_{hnd2,f,ip2} = E_{hnd2} - | E_{Cload} | - E_{hnd2,ip2r}$ (5.11)

6. Note that, as stated in **3**, the quantities $E_{hnd2,ip1r}$, $E_{hnd2,ip1f}$, $E_{hnd2,ip2r}$, and $E_{hnd2,ip2f}$ used in (4) and (5) could simply be approximated as 0 here.

5. 2. Synopsys Model of Power Dissipation

Various power estimation tools, including the one provided in Synopsys's Design Analyzer®, calculate power using models based on some modification of the following algorithm:

1. Instead of directly calculating power itself, the tool computes the amount of energy dissipated within the simulation duration, and then divides the computed amount of energy with the simulation time.
2. For static power calculation, a cell is characterized by its static power dissipation, either average or for a specific state (e.g. for output = 0 and for output = 1). Then, its static power dissipation is calculated by dividing static energy dissipation with simulation time. For example, if a NAND gate, over a simulation course of 110 ns, is on (output = 1) for 70 ns and on (output = 0) for the other 40 ns, we have:
   average static power = $(70 \times P_{static,output=High} + 30 \times P_{static,output=Low}) / 110$

3. For dynamic energy, cells are characterized by two things, namely: first, input/output capacitance, which is also used for timing calculations; and second, by the amount energy dissipated on logic transition on each pins on rise/fall transition when other pins are at a certain logic value, or on rise/fall transition independent of logic states on other pins, or simply on transition.
4. If an input pin of a cell experiences transition, it is assumed to dissipate a specified amount of internal energy (not power). If this amount of energy is not specified, use some default. If even this default is not specified, assume to be zero. If this pin experiences many transitions during the simulation period, sum the contributions of all those transitions (which may or may not be all identical – as it may depend on the states of some other pins as well).
5. If an output pin of a cell experiences transition, it is assumed to dissipate a specified amount of internal energy (not power). If this amount of energy is not specified, use some default. If even this default is not specified, assume to be zero. If this pin experiences many transitions during the simulation period, sum the contributions of all those transitions (which may or may not be all identical – as it may depend on the states of some other pins as well). Additionally, any capacitance connected to the output pin may switch, and the contribution of switching energy (= $\frac{1}{2}V_{DD}^2 \times \Sigma C$) should also be added in like manner.
6. If there are many individual cells (gates, flip-flops, etc) in the circuit, repeat the above steps for all cells and sum the result together for the entire simulation period.
7. To obtain power dissipation (average), just divide the dissipated energy with simulation time.

Actually, the algorithms used by power estimation tools in Design Analyzer® (or any other power estimation tools) are necessarily more complicated than the aforementioned algorithm, as those tools need to present the breakdown of power, for example, how many percent occur in this part of the design, how many percent due to leakage power, etc. However, the significant thins is that in characterizing a cell's power dissipation for use in such tools, we need to measure the following quantities:

1. Leakage power for each state, as well as (if a simpler model is desired) average for high / low output, or even average for all possible states, assuming some probabilities for each state (unless known to be otherwise, assume that all possible states are equally likely).

2. Amount of internal energy dissipated for rising or falling logic transition at every pin of the cell. Note that this depends on some other factors as well, such as the *transition delay* of the input waveform and the *load capacitance* driven by the output pins of the cells. For output pins, if the logic value at an output pin can change due to transitions at several different input pins, separate measurements have to be made for the output transitions at each of the input pins, and an average will need to be obtained as well if a simpler Synopsys model is desired.

It is not necessary to find switching energy of the load capacitance (except to find internal energy), as this will be calculated on power estimation time based on the load capacitance (it is assumed that the cells are already characterized for their input capacitance, as well as for their other timing parameters).

Further, note that the internal energy per transition (henceforth referred to simply as ***"internal energy"***) associated with a pin for a given cell is not a constant, but depends on, among others:

- load conductance being driven by the cell
- shape of input waveform causing the internal energy dissipation

In Synopsys models for MOS, load conductance is represented by load capacitance, while the shape of input waveform is quantified by input ***transition delay***, namely the portion of the propagation delay of the driving cell which occurs since the driving cell has to drive capacitive load (***see the chapter on timing and capacitance characterization***). Since the relation between internal power and load capacitance / input transition delay is not necessarily well characterized by equations, the model is simply in the form of a lookup table, which may be one-dimensional or two dimensional.

- in the one-dimensional model, the power is indexed against various values of ***input waveform transition delay (for input pins)*** which may differ between rising and falling transition delay, and against various values of load capacitances for output pins.
- in the two-dimensional model, the power is indexed against both input transition delay and load capacitance (using 2-D matrices).

Regarding point (1) namely that actually it is energy and not power which is found first, here let us use an example. Returning to **Figure 5.2**, and let us suppose that the waveforms had arisen from a simulation; we then have the followings:

- Simulation runtime = 18ns
- Transitions on pin ip1: 4 transitions (2 rise, 2 fall)
- Transitions on pin ip2: 4 transitions (2 rise, 2 fall)
- Transitions on pin op: 4 4 transitions (2 rise, 2 fall)
- Capacitive load on pin op sees 4 transitions (whether rising or falling is immaterial)

In the following example, dynamic power will be computed. Let us suppose we have two modeling alternatives for internal energy estimation:

Alternative 1:

- pin ip1 is characterized by rise energy and fall energy; let us called them er1 and ef1, respectively
- pin ip2 is characterized by rise energy when ip1=0, rise energy when ip1=1, fall energy when ip1=0, and fall energy when ip1=1; let us denote them as er2_0, er2_1, ef2_0, and ef2_1, respectively.
- pin op is characterized by rise and fall energy; let us denote them as eopr and eopf, respectively

Therefore:

Internal power = total internal energy / observation duration

= (ip1 internal energy + ip2 internal energy + op internal energy) / 18ns

= [(2×er1 + 2×ef1) + (1×er2_0 + 1×er2_1 + 1×ef2_0 + 1×ef2_1) + (2×eopr + 2×eopf)] / 18ns

Switching power = $4 \times \frac{1}{2} C_{load} V_{dd}^2$ / observation duration

Alternative 2:

- pin op only is characterized by average energy per transition; let us call this eop

Therefore:

Internal power	=	total internal energy / 18ns
	=	(ip1 internal energy + ip2 internal energy + op internal energy) / 18ns
	=	[0 + 0 + 4×eop] / 18ns
Switching power	=	$4 \times \frac{1}{2} C_{load} V_{dd}^2$ / 18ns

Note that *alternative 2* is much simpler, yet in our case is practically just as accurate as *alternative 1*, since for simple combinational cell input transition does not cause much power dissipation unless it causes output transition (and therefore all internal energy parameters could simply be lumped to output pins), and also because for our (apparently sufficiently long) observation period, the number of rising transitions on output pin *op* are equal. However, these may not be true for more complicated cells, which may exhibit internal switching which may not propagate to output and hence their input energies are not negligible, or for shorter observation periods, in which the number of rising and falling transitions may be unequal.

Note also that regardless of the modeling alternative chosen, the amount of switching power dissipated should always be the same.

5. 3. Proposed Power Characterization Method

5. 3. 1. Static Power Measurement and Modeling

5. 3. 1. 1. Basic Method

The proposed modeling alternative for static power characterization of relatively simple cells is by *assigning one value for each possible output value* for each cell. For example, for a full adder cell, which has two output pins (output and carry-out) and therefore four possible output values, we should have four possible leakage power values. This may not exhaustively enumerate all possible values (as there are eight possible input combinations), but should be adequate nonetheless. Note, however, that *for simple combinational circuits, all possible input*

combinations should be tried, although the output value used is only one for each possible output. For sequential cell, this does not apply, and instead only the most likely input sequence (i.e. “normal sequence” instead of reset for a flip-flop).

Most relatively simple cells in fact either one output pin only, or a pair of complementary output pins (such as D flip-flop with both *q* and *qb* output), and therefore only two possible output value.

The method for leakage power measurement is by simply forcing the cell to the desired output value, and then *after sufficiently long (10 ns after the stimulus is performed should be enough), after all output transients die down, the measurement is performed*. Note that *for sequential cells, for static power measurements purposes, the intended state must be reached in “normal” way*, e.g. for a flip-flop, must be attained through application of input and clock instead of asynchronous reset and set.

5. 3. 1. 2. SPICE Example

As an example, the following fragment of SPICE code characterized the leakage power of a 2-input NAND gate. This is performed by instantiating four NAND gates, each fed permanently with one input combination, and then use .measure statement to measure power dissipation of each NAND gate (each case). Note that here the four power dissipations are named `pstat00h`, `pstat01h`, `pstat10h`, and `pstat11l`, respectively. Note also that the final value is the last two (one each for high and low output) instead of the four aforementioned values. The NAND gate used here is `hnd2` from the 0.5µm library.

```
.param spl='3.3v' ** Use 3.3V Power Supply
vspstat spstat gnd spl

xstat00h spstat gnd gnd op00h hnd2        ** For input = 00
xstat01h spstat gnd spstat op01h hnd2     ** For input = 01
xstat10h spstat spstat gnd op10h hnd2     ** For input = 10
xstat11l spstat spstat spstat op11l hnd2  ** For input = 11

.measure pstat00h avg p(xstat00h) from=10ns to=11ns  ** These time intervals
.measure pstat01h avg p(xstat01h) from=10ns to=11ns  ** are arbitrary for
```

```
.measure pstat10h avg p(xstat10h) from=10ns to=11ns  ** combinational cells
.measure pstat11l avg p(xstat11l) from=10ns to=11ns  ** here we use 1 ns

.measure pstat_av param='(pstat00h+pstat01h+pstat10h+pstat11l)/4'
.measure pstat_avl param='(pstat00h+pstat01h+pstat10h)/3'
.measure pstat_avh param='pstat11l'

.tran 0.01ns 18ns
```

When the above lines were simulated, the results (in Watts) were as follows:

```
  pstat00h=  3.2004E-13  from=  1.0000E-08     to=  1.1000E-08
  pstat01h=  2.6363E-12  from=  1.0000E-08     to=  1.1000E-08
  pstat10h=  6.4790E-12  from=  1.0000E-08     to=  1.1000E-08
  pstat11l=  3.2261E-13  from=  1.0000E-08     to=  1.1000E-08
  pstat_av=  2.4395E-12
  pstat_avl=  3.1451E-12  ** this is the final result for high output
  pstat_avh=  3.2261E-13  ** and this is for low output
```

5. 3. 2. Dynamic Power Measurement for Simple Combinational Cells

5. 3. 2. 1. Basic Method

For simple combinational cells, *it is proposed that internal energy dissipation of cells be characterized by average rising output energy and average falling output energy*, assuming that only one input changes, *modeled as a function of load capacitance only*, and that all inputs which could change the output are equally likely to change. For example, for a 3-input NAND gate, if two inputs are held at high (the non-controlling value), the other one input could change the output. This is to be performed for each of those three inputs: the input is changed when the other two are held at non-controlling value. At the end, we will have three values of energy associated with rising output (one associated with the change in each input pin, respectively). Then, they are simply averaged. The same steps are also to be performed with energy associated with falling output. Note that here we ignore the possibility of several inputs changing at once. This should cause only a limited loss of accuracy, if at all.

Also, the stimulus waveform should be representative. A waveform degrading circuit should be used. This circuit should consist of an cell of the same type, singly or in chain, driving a load in addition to the cell being tested. This should generate a more representative input slope.

5. 3. 2. 2. SPICE Example

The following example applies the abovementioned method to the 3-input NAND. To measure the internal energies, we could simply instantiate the circuit in Figure 5. 3 and apply the stimulus in Figure 5. 4, as in the following example. In this example, the internal energies of ip1 and ip2 are simply approximated as 0 as in point (3) and hence not measured, while the integration interval of power (to obtain energy) is somewhat arbitrarily made to last 2 ns long, starting from the start of input transition to 2 ns later. The NAND gate used is hnd2 from the 0.5μm library. Note that this is just an example – in actual measurement a different circuit and a different measurement interval will be used.

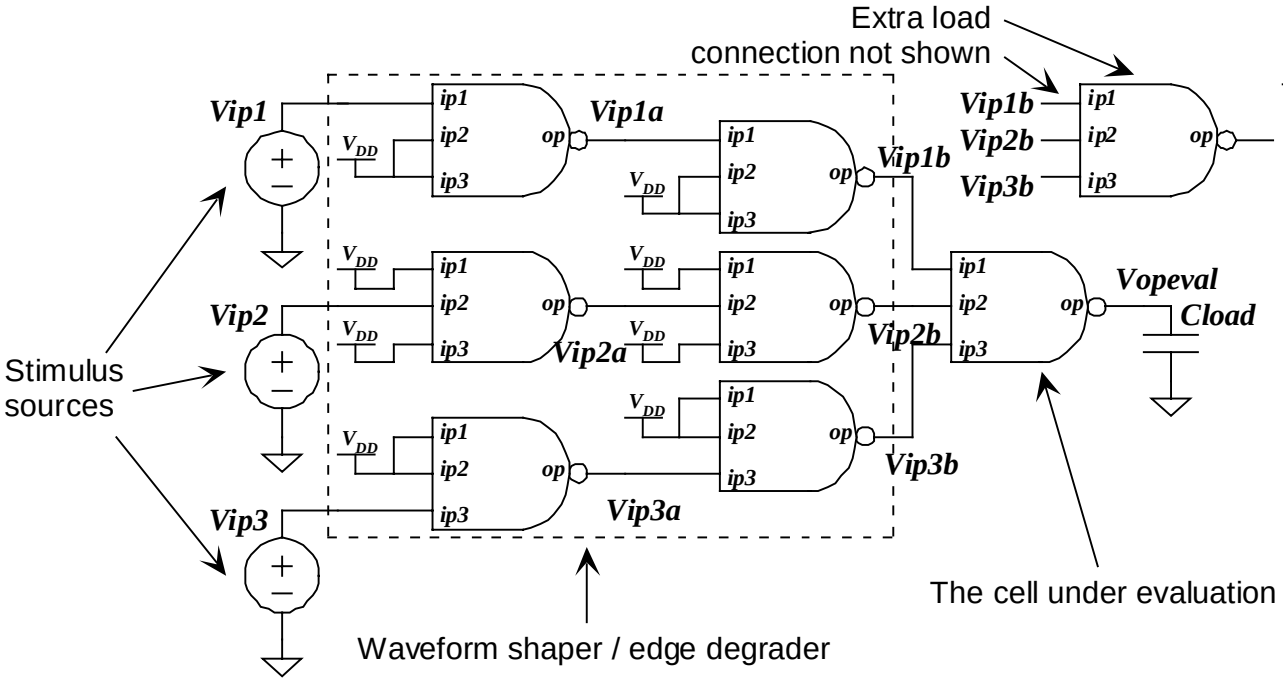


Figure 5. 3. hnd3 Power Characterization Circuit

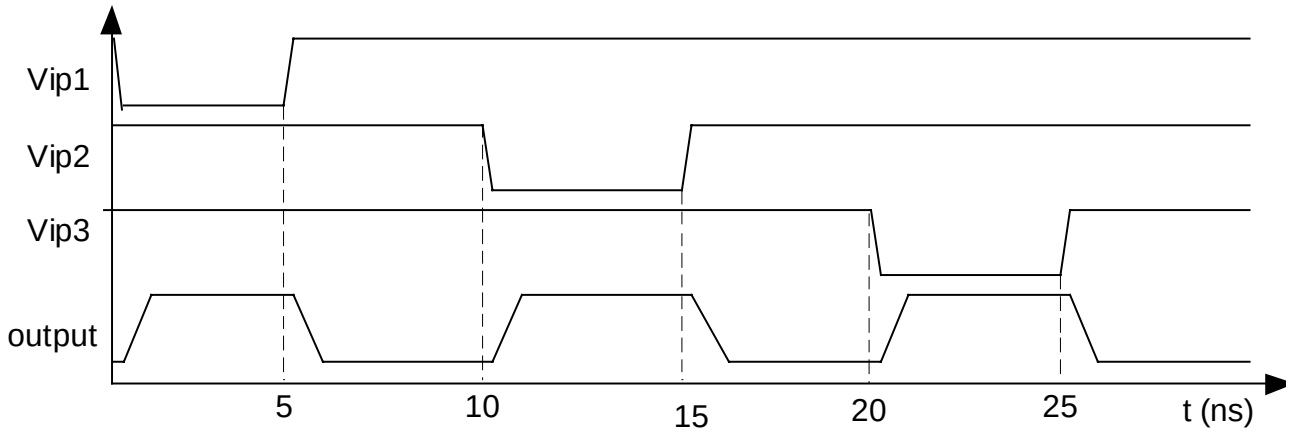


Figure 5. 4. Waveforms Used for hnd3 Power Characterization

Note again that internal energy of output pin is modeled as variable of load capacitance only. Therefore, the slope of input waveform does not have to be well defined and, more importantly, does not have to be the same for all cells, which in turn means that the load capacitance driven by the cell driving the cell being evaluated (edge degraders) need not be well defined. For that reason, they all drive the next stage directly here instead of via dependent sources.

Note that the internal energy of the output pin is measured for eight different values of load capacitance (in 0fF – 500fF range). It does not seem likely that this type of NAND gate will have to drive much more than 10-20 loads, so this range seems adequate.

At any case, SPICE finds energy by integrating power dissipation with respect to time. Here the measurement performed are:

Output Event	*Due to*	*Power integration period (see.measure statements)*
rise	ip1	0 – 5 ns
fall	ip1	5 – 10 ns
rise	ip2	10 – 15 ns
fall	ip2	15 – 20 ns
rise	ip3	20 – 25 ns
fall	ip3	25 – 30 ns

```
hnd3 Power Characterization

.inc hivsp

.param vspl='3.3v'
vdd dd gnd vspl

xhnd3buf1a dd ip1 dd dd ip1a hnd3
xhnd3buf1b dd ip1a dd dd ip1b hnd3

xhnd3buf2a dd dd ip2 dd ip2a hnd3
xhnd3buf2b dd dd ip2a dd ip2b hnd3

xhnd3buf3a dd dd dd ip3 ip3a hnd3
xhnd3buf3b dd dd dd ip3a ip3b hnd3

xload1 dd ip1b ip2b ip3b opld hnd3
xeval dd ip1b ip2b ip3b opeval hnd3

.param loadc='0fF'

cload opeval gnd loadc

xleakh dd gnd gnd gnd opleakh hnd3
xleakl dd dd dd dd opleakl hnd3

vip1 ip1 gnd pwl (0ns vspl 0.1ns 0 5ns 0 5.2ns vspl
+ 30ns vspl)
vip2 ip2 gnd pwl (0ns vspl 10ns vspl 10.1ns 0 15ns 0 15.2ns vspl
+ 30ns vspl)
vip3 ip3 gnd pwl (0ns vspl 20ns vspl 20.1ns 0 25ns 0 25.2ns vspl 30ns vspl)

.measure pleakh avg p(xleakh) from=0ns to=1ns
.measure pleakl avg p(xleakl) from=0ns to=1ns
.measure eleakh integral p(xleakh) from=0ns to=3ns
.measure eleakl integral p(xleakl) from=0ns to=3ns

.measure eswrise param='0.5*loadc*vspl*vspl'
.measure eswfall param='eswrise'

** Measurement of ip1 -> op energies

.measure edynrise1 integral p(xeval) from=5ns to=10ns
.measure edynfall1 integral p(xeval) from=0ns to=5ns

.measure eintrise1 param='edynrise1-eswrise'
.measure eintfall1 param='edynfall1-eswfall'

** Measurement of ip2 -> op energies

.measure edynrise2 integral p(xeval) from=15ns to=20ns
.measure edynfall2 integral p(xeval) from=10ns to=15ns

.measure eintrise2 param='edynrise2-eswrise'
.measure eintfall2 param='edynfall2-eswfall'
```

```
** Measurement of ip3 -> op energies

.measure edynrise3 integral p(xeval) from=25ns to=30ns
.measure edynfall3 integral p(xeval) from=20ns to=25ns

.measure eintrise3 param='edynrise3-eswrise'
.measure eintfall3 param='edynfall3-eswfall'

.measure eintriser param='(eintrise1+eintrise2+eintrise3)/3.0'
.measure eintfaller param='(eintfall1+eintfall2+eintfall3)/3.0'

** Finding the relative portions of each type of energy
.measure eleakav param='0.5*(eleakh+eleakl)'
.measure eswav param='0.5*(eswrise+eswfall)'
.measure einav
param='(eintrise1+eintfall1+eintrise2+eintfall2+eintrise3+eintfall3)/6'
.measure edynav param='eswav+einav'
.measure etotav param='edynav+eleakav'
.measure peleak param='100*eleakav/etotav'
.measure pedyn param='100*edynav/etotav'
.measure pesw param='100*eswav/etotav'
.measure pein param='100*einav/etotav'

.op
.option post nomod accurate
.option converge=1 gmindc=1e-12
.tran 0.025ns 32ns

.alter 1: for 25fF load:
.param loadc='25fF'
.alter 2: for 50fF load:
.param loadc='50fF'
.alter 3: for 75fF load:
.param loadc='75fF'
.alter 4: for 100fF load:
.param loadc='100fF'
.alter 5: for 150fF load:
.param loadc='150fF'
.alter 6: for 250fF load:
.param loadc='250fF'
.alter 7: for 500fF load:
.param loadc='500fF'

.end
```

When the above lines were simulated, the output file contained the following lines, among others. Note that the results for internal power were energies (not power) in Joules.

```
.
** Leakage Power in W **
  pleakh=  2.9351E-14  from=  0.0000E+00     to=  1.0000E-09
  pleakl=  6.4523E-13  from=  0.0000E+00     to=  1.0000E-09
```

```
.
.
** Internal Energy (Rise and Fall Energies, respectively) of output pin **
.
  eintriser=  4.8693E-13   ** for Cload = 0fF
  eintfaller=  5.9903E-13
.
.
  eintriser=  4.9262E-13   ** for Cload = 25fF
  eintfaller=  5.9781E-13
.
.
  eintriser=  5.9759E-13   ** for Cload = 50fF
  eintfaller=  5.9678E-13
.
.
  eintriser=  5.0171E-13   ** for Cload = 75fF
  eintfaller=  5.9587E-13
.
.
  eintriser=  5.0486E-13   ** for Cload = 100fF
  eintfaller=  5.9593E-13
.
.
  eintriser=  5.0913E-13   ** for Cload = 150fF
  eintfaller=  5.9596E-13
.
.
  eintriser=  5.1439E-13   ** for Cload = 250fF
  eintfaller=  5.9613E-13
.
.
  eintriser=  5.2007E-13   ** for Cload = 500fF
  eintfaller=  5.9413E-13.
.
.
```

5. 3. 3. Dynamic Power Measurement for Sequential Cells and Combinational Cells with Internal Loads

5. 3. 3. 1. Basic Method

As previously stated in previous discussions on simple combinational cells, we could assume that unless a logic transition at input causes a logic transition at the cell’s output, the amount of internal power / energy dissipated will be relatively negligible and could then simply be assumed to be 0. However, this is not true of combinational cells with internal loads or of

sequential circuits, both of which may incur significant power dissipation due to input transitions even if no logic transitions occur at primary outputs as a result.

As a simple example, let us use a hypothetical 4-input AND-OR (actually, NAND-NAND) gate consisting of three NAND gate as in the following Figure 5. 5 :

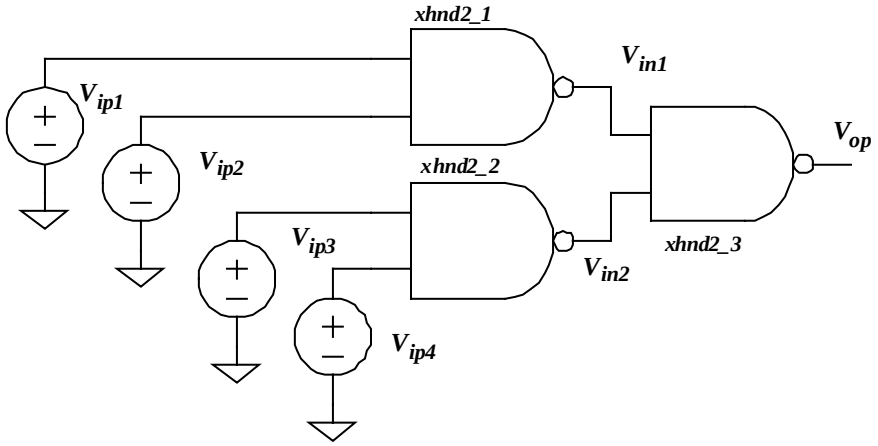


Figure 5. 5. A 4-input AND-OR gate

Suppose we apply the stimuli shown in Figure 5.6, and as a result obtain the following output and power waveform as shown:

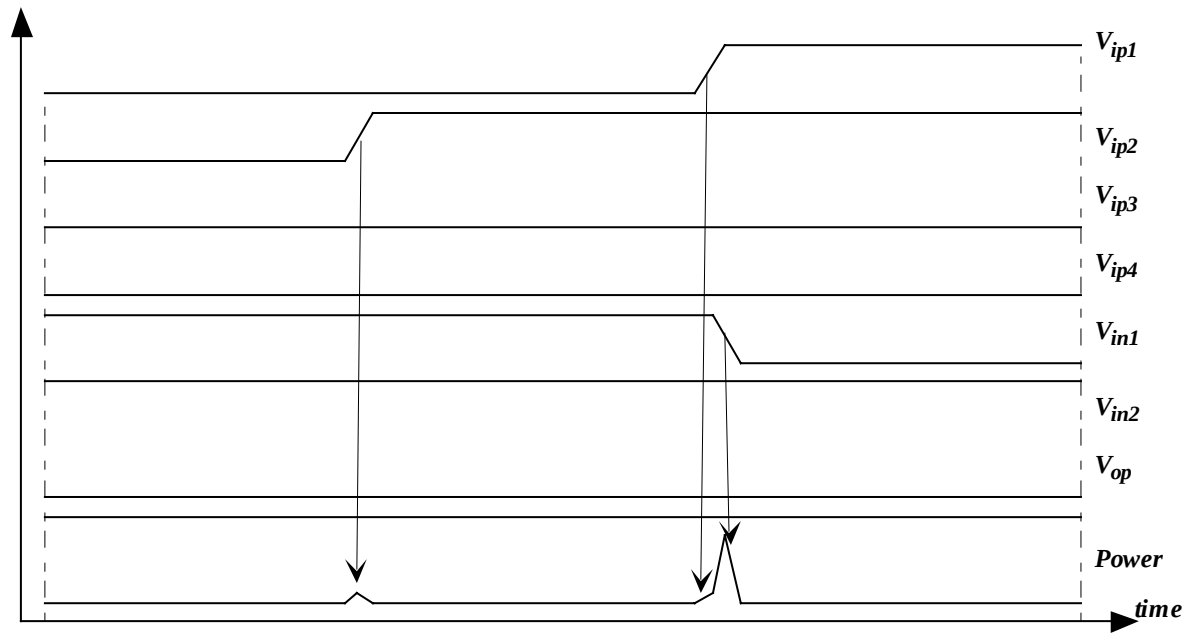


Figure 5. 6. Input stimuli (V_{ip1} through V_{ip4}) and circuit response. Note that while the first input transition (at ip2) does not cause any significant power dissipation by the AND-OR gate, the second transition (at ip1) does.

As seen in Figure 5.6, there are two transitions at input, neither of which causes logic transitions at output. However, those two transitions cause significantly different amount of energy dissipation. The **rising transition at ip2** (the earlier one of the two transitions) not only does not cause logic transitions at the primary output op, it in fact does not even cause logic transitions at intermediary nodes in1 and in2. Therefore, while some glitch may occur at intermediary nodes and primary output, the resulting dynamic energy dissipation should be negligible. However, the latter one of the two input transition, namely the **rising transition at ip1**, causes a logic transition at an internal load (falling transition at in1). The upper left NAND gate, as a result of the input transition, exhibits an output transition and hence substantial energy dissipation. Therefore, although the AND-OR gate does not exhibit any logic transition at its output, a substantial energy dissipation occurs. This energy dissipation cannot be considered zero. Hence, for all pins (inputs and outputs), internal energy must be measured and cannot simply be considered zero.

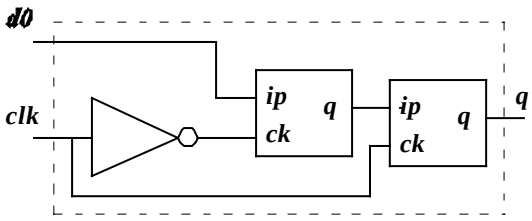


Figure 5. 7. A positive-edge triggered master-slave D flip-flop

Similarly, for sequential cells, some input transitions may exist which would cause similar logic transition at some internal loads, and hence cause significant power dissipation, without necessarily causing logic transition at output. For example, in the D flip-flop at Figure 5.7, an input change when clk = low will cause logic transition at the output of the first D latch, and dissipates some internal energy, although that alone (without further change in clk) would not cause output change.

In general, it is not possible to predict which cases of input change will or will not dissipate energy, and therefore which cases to consider, without knowing the internal structure of the cell.

Since any input transition may cause significant energy dissipation, both output- and input-pin related internal energies must be measured, as follows:

- **OUTPUT PINS:** modeled as a function of load capacitance. The input slope is generated using the smallest cell in the library – typically, the inverter – driving a “typical” load.
- **INPUT PINS:** modeled as a function of input transition delay, with output pins driving a “typical” load.

Note that transition delay must be defined or measurable unambiguously. Additionally, the transition delay of the input waveform should be the same for all cells in the library whenever possible – and this demands that the load driven by the driving gate is always made the same for all sequential cells (or combinational cells with internal load) tested. For the latter

reason, it is proposed that the connection between the driving gates and the cells being evaluated vis made indirectly via dependent sources, so that capacitance values could be specified in the simulation file.

5. 3. 2. 2. SPICE Example

As an example, here an example SPICE file for characterizing the D flip-flop `hdpq` from the 0.5µm library is presented. The schematic for the characterization circuit is as shown in Figure 5.8 (next page)

It appears that `hdpq` is a master-slave flip-flop similar to the one previously shown in Figure 5. 7. Hence, if input transition at `ip` occurs when ck = high, it would not cause a substantial internal energy dissipation, since at that time the first high-active latch stage is inactive. However, if the ip input changes wen clk=low, then a significant internal energy dissipation would occur, since at that time the first latch stage is active.

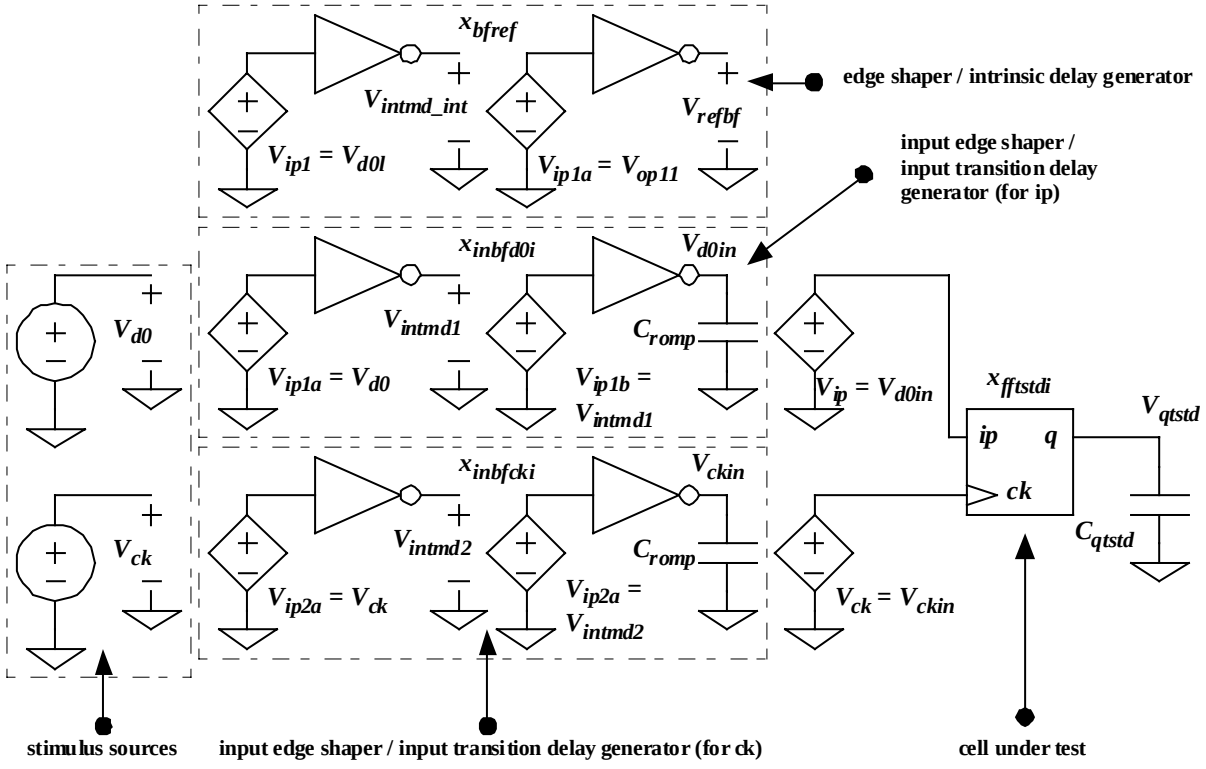


Figure 5. 8. Circuit for Power Measurement for `hdpq`

Further, if the input `ip` changes when clk = high, such that ip and the output q of the flip-flop are different, this in itself would not cause significant internal energy dissipation. However, after the *falling* edge of the clock comes, the first latch stage will suddenly becomes active, and it will respond to the already changed `ip` value accordingly; the falling clock edge, therefore, will cause the output of the first latch to change. Hence, the dissipated energy will be much greater than dissipated by falling edges of clock which come when input = output. Therefore, they need to be measured separately.

Consequently, the following waveforms had been used in power measurements (see Figure 5. 9)

Note that input Vd0 changes at 12, 44, 52, and 68 ns, respectively. As in measurement for hnd2 power dissipation, a 3 ns integration interval is used. The measurements are:

Measurement	From (ns)	To (ns)
Internal energy due to d0 falls, clock low	44	47
Internal energy due to d0 rises, clock low	12	15
Internal energy due to clk falls, ip = 1, q = 1 (will not change output)	8	11
Internal energy due to clk falls, ip = 0, q = 0 (will not change output)	40	43
Internal energy due to clk falls, ip = 0, q = 1 (will change output after rising edge)	56	59
Internal energy due to clk falls, ip = 1, q = 0 (will change output after rising edge)	72	75
Internal energy due to clk rises, output low and unchanged, normal operation	32	35
Internal energy due to clk rises, output high and unchanged, normal operation	0	3
Internal energy due to q rises	16	19
Internal energy due to q falls	48	51

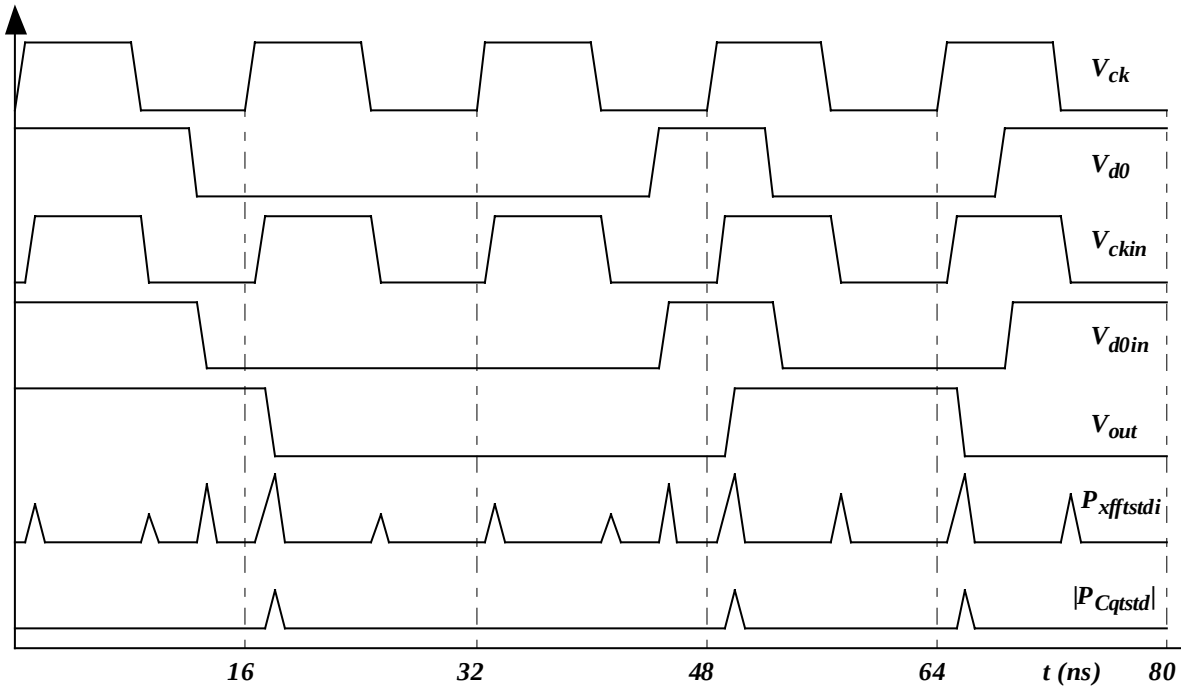


Figure 5. 9. Waveforms used in Power Measurements for `hdpq`

Note that since the internal energy depend on load capacitance, several different values of input slope and / or output capacitance are used, namely:

For internal energies associated with output pins:

Input slope: created with C_{romp} = 75fF

Load capacitance: ALTERed among 0fF, 25fF, 50fF, 75fF, 100fF, 150fF, 250fF, 500fF

For internal energies associated with input pins:

Input slope: ALTERed among C_{romp} = 0fF, 35fF, 75fF, 150fF, 500fF, 1000fF

Load capacitance: 75fF

The SPICE file used was as follows (`hdpq_pwr.sp`):

```
hdpq Power Dissipation Parameters Characterization

.lib "hp05um_model" cmos_models
.inc hiv.sp
```

```
.inc hdpq.sp

** Subcircuit definitions
.subckt noninvbuf dd ip op
xinv dd ip intmd loadfreehiv
xbck dd intmd op loadfreehiv
.ends noninvbuf

.subckt nonloadhdpq dd d0 ck q
ed0 d0in gnd d0 gnd 1
eck ckin gnd ck gnd 1
xff dd d0in ckin q hdpq
.ends nonloadhdpq

.param vspl='3.3v'
vdd dd gnd vspl

** Leakage power measurement **
xffleak ddd d0l ckl opl hdpq
vd0l d0l gnd 0
vckl ckl gnd pwl (0ns 0 1ns 0 1.2ns vspl)
.measure plkh avg p(xffleak) from=0ns to=1ns    ** leakage power, q = H
.measure plkl avg p(xffleak) from=55ns to=56ns  ** leakage power, q = L

** Dynamic power measurement **
** Intrinsic input delay measurements - needed for characterizing some input
**
** power which may may depend on input slope **
xbfref dd ck refbf noninvbuf
.measure intrise trig v(ck) val='0.5*vspl' rise=1 targ v(refbf)
val='0.5*vspl'
+ rise=1
.measure intfall trig v(ck) val='0.5*vspl' fall=1 targ v(refbf)
val='0.5*vspl'
+ fall=1

xinbfd0i dd d0 d0in noninvbuf
.param d0c='0fF'
cd0 d0in gnd d0c
.measure d0trise trig v(d0) val='0.5*vspl' rise=1 targ v(d0in) val='0.5*vspl'
+ rise=1
.measure d0tfall trig v(d0) val='0.5*vspl' fall=1 targ v(d0in) val='0.5*vspl'
+ fall=1
.measure d0_dtr param='d0trise-intrise'
.measure d0_dtf param='d0tfall-intfall'

xinbfcki dd ck ckin noninvbuf
.param ckc='0fF'
cck ckin gnd ckc
.measure cktrise trig v(ck) val='0.5*vspl' rise=1 targ v(ckin) val='0.5*vspl'
+ rise=1
.measure cktfall trig v(ck) val='0.5*vspl' fall=1 targ v(ckin) val='0.5*vspl'
+ fall=1
.measure ck_dtr param='cktrise-intrise'
.measure ck_dtf param='cktfall-intfall'

xfftstdi dd d0in ckin qtstdi nonloadhdpq
```

```
.param qtstdic='0fF'
cqtstd qtstdi gnd qtstdic

vck ck gnd pulse (0v vspl 0ns 0.4ns 0.4ns 7.6ns 16ns)
vd0 d0 gnd pwl (0ns vspl 12ns vspl 12.4ns 0v 44ns 0v 44.4ns vspl 52ns vspl
+ 52.4ns 0v 68ns 0v 68.4ns vspl)

** Measuring internal energy due to d0 transition: **
** For rising d0 ... **
.measure eid0up integral p(xfftstdi.xff) from=12ns to=15ns
** and for falling d0 ... **
.measure eid0dn integral p(xfftstdi.xff) from=44ns to=47ns

** Measuring internal energy due to falling CLK edge: **
** For input = output : **
** For Output = H and Input = H... **
.measure eickfall_11 integral p(xfftstdi.xff) from=8ns to=11ns
** and for Output = L and Input = L... **
.measure eickfall_00 integral p(xfftstdi.xff) from=40ns to=43ns
** And the average of the (input = output) case ...
.measure eickfall_qc param='0.5*(eickfall_11+eickfall_00)'
** For input != output : **
** For Output = H and Input = L... **
.measure eickfall_10 integral p(xfftstdi.xff) from=56ns to=59ns
** and for Output = L and Input = H... **
.measure eickfall_01 integral p(xfftstdi.xff) from=72ns to=75ns
** and the average for (input != output) case ...
.measure eickfall_qt param='0.5*(eickfall_10+eickfall_01)'

** Measuring internal energy due to rising CLK without output change: **
** For Output = H ... **
.measure eickrise_qh integral p(xfftstdi.xff) from=0ns to=3ns
** and for Output = L ... **
.measure eickrise_ql integral p(xfftstdi.xff) from=32ns to=35ns
** and their average ..
.measure eickrise_av param='0.5*(eickrise_qh+eickrise_ql)'

** And (finally!) measuring internal power due to output transitions **
** First, switching energy per toggle **
.measure eswrise param='0.5*qtstdic*vspl*vspl'
.measure eswfall param='eswrise'
** Then measure the internal energy, for rising output ... **
.measure etot_qrise integral p(xfftstdi.xff) from=16ns to=19ns
.measure ei_qrise param='etot_qrise-eswrise-eickrise_ql'
** and for falling output ... **
.measure etot_qfall integral p(xfftstdi.xff) from=48ns to=51ns
.measure ei_qfall param='etot_qfall-eswfall-eickrise_qh'

** Simulation Directives **
.op
.option post nomod accurate
.option gmindc=1e-12
.tran 0.1ns 80ns

** ALTER statements to see the effects of transition time on internal power
** consumption. It was decided that internal power was more affected by
** input transition time (here represented by load capacitances for the
```

```
** previous stage) than by load capacitance. The output load of the FF
** was taken to be 75 fF - it seemed sensible, that's why.

.alter Load of Previous Stage = 0 fF
.param ckc='0fF'
.param d0c='0fF'
.param qtstdic='75fF'
.alter Load of Previous Stage = 35 fF
.param ckc='35fF'
.param d0c='35fF'
.param qtstdic='75fF'
.alter Load of Previous Stage = 75 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='75fF'
.alter Load of Previous Stage = 150 fF
.param ckc='150fF'
.param d0c='150fF'
.param qtstdic='75fF'
.alter Load of Previous Stage = 500 fF
.param ckc='500fF'
.param d0c='500fF'
.param qtstdic='75fF'
.alter Load of Previous Stage = 1000 fF
.param ckc='1000fF'
.param d0c='1000fF'
.param qtstdic='75fF'

** ALTER statement for Q energy determination
.alter Load = 0 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='0fF'
.alter Load = 25 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='25fF'
.alter Load = 50 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='50fF'
.alter Load = 75 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='75fF'
.alter Load = 100 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='100fF'
.alter Load = 150 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='150fF'
.alter Load = 250 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='250fF'
```

```
.alter Load = 500 fF
.param ckc='75fF'
.param d0c='75fF'
.param qtstdic='500fF'

.end
```

When the above SPICE file was simulated, the output contains the following lines, among others:

```
.
.
.
******
 load of previous stage = 0 ff
 ******  transient analysis               tnom=  25.000 temp=  25.000

[Results for Cromp = 0fF, load of ff = 75fF, for measurement of input-pin-
associated internal energies]

******
  plkh            =  9.7683E-49  from=  0.0000E+00     to=  1.0000E-09
  plkl            =  1.9667E-12  from=  5.5000E-08     to=  5.6000E-08
  intrise         =  1.0308E-10  targ=  3.0308E-10   trig=  2.0000E-10
  intfall         =  1.5474E-10  targ=  8.3547E-09   trig=  8.2000E-09
  d0trise         =  1.0346E-10  targ=  4.4303E-08   trig=  4.4200E-08
  d0tfall         =  1.4757E-10  targ=  1.2348E-08   trig=  1.2200E-08
  d0_dtr          =  3.8077E-13
  d0_dtf          = -7.1639E-12
  cktrise         =  1.0308E-10  targ=  3.0308E-10   trig=  2.0000E-10
  cktfall         =  1.5474E-10  targ=  8.3547E-09   trig=  8.2000E-09
  ck_dtr          =  0.0000E+00  ** Transition delay of buffer = 0, since
  ck_dtf          =  0.0000E+00  ** Cromp = 0fF
  eid0up          =  1.7420E-12  from=  1.2000E-08     to=  1.5000E-08
  eid0dn          =  1.6908E-12  from=  4.4000E-08     to=  4.7000E-08
  eickfall_11     =  5.9958E-13  from=  8.0000E-09     to=  1.1000E-08
  eickfall_00     =  6.7957E-13  from=  4.0000E-08     to=  4.3000E-08
  eickfall_qc     =  6.3957E-13
  eickfall_10     =  2.1418E-12  from=  5.6000E-08     to=  5.9000E-08
  eickfall_01     =  2.3526E-12  from=  7.2000E-08     to=  7.5000E-08
  eickfall_qt     =  2.2472E-12
  eickrise_qh     =  7.1355E-13  from=  0.0000E+00     to=  3.0000E-09
  eickrise_ql     =  6.6549E-13  from=  3.2000E-08     to=  3.5000E-08
  eickrise_av     =  6.8952E-13
  eswrise         =  4.0837E-13
  eswfall         =  4.0837E-13
  etot_qrise      =  2.4707E-12  from=  1.6000E-08     to=  1.9000E-08
  ei_qrise        =  1.3969E-12
  etot_qfall      =  2.3577E-12  from=  4.8000E-08     to=  5.1000E-08
  ei_qfall        =  1.2358E-12

         ***** job concluded
******  Star-HSPICE  --    98.2  (980711)  16:14:46  01/07/2000  solaris
```

```
******
 load of previous stage = 0 ff
 ******  job statistics summary           tnom=  25.000 temp=  25.000
.
.
.
******
 load = 100 ff

[Results for Cromp = 75fF, load = 100fF, for measurement of output-pin-
associated internal energies]

 ******  transient analysis               tnom=  25.000 temp=  25.000
******
  plkh            =  9.7683E-49  from=  0.0000E+00     to=  1.0000E-09
  plkl            =  1.9666E-12  from=  5.5000E-08     to=  5.6000E-08
  intrise         =  1.0308E-10  targ=  3.0308E-10   trig=  2.0000E-10
  intfall         =  1.4668E-10  targ=  8.3467E-09   trig=  8.2000E-09
  d0trise         =  1.8102E-10  targ=  4.4381E-08   trig=  4.4200E-08
  d0tfall         =  1.9654E-10  targ=  1.2397E-08   trig=  1.2200E-08
  d0_dtr          =  7.7945E-11
  d0_dtf          =  4.9856E-11
  cktrise         =  1.8115E-10  targ=  3.8115E-10   trig=  2.0000E-10
  cktfall         =  1.9793E-10  targ=  8.3979E-09   trig=  8.2000E-09
  ck_dtr          =  7.8076E-11
  ck_dtf          =  5.1250E-11
  eid0up          =  1.6878E-12  from=  1.2000E-08     to=  1.5000E-08
  eid0dn          =  1.6694E-12  from=  4.4000E-08     to=  4.7000E-08
  eickfall_11     =  5.6983E-13  from=  8.0000E-09     to=  1.1000E-08
  eickfall_00     =  6.6026E-13  from=  4.0000E-08     to=  4.3000E-08
  eickfall_qc     =  6.1505E-13
  eickfall_10     =  2.1077E-12  from=  5.6000E-08     to=  5.9000E-08
  eickfall_01     =  2.2977E-12  from=  7.2000E-08     to=  7.5000E-08
  eickfall_qt     =  2.2027E-12
  eickrise_qh     =  6.8510E-13  from=  0.0000E+00     to=  3.0000E-09
  eickrise_ql     =  6.2537E-13  from=  3.2000E-08     to=  3.5000E-08
  eickrise_av     =  6.5523E-13
  eswrise         =  5.4450E-13
  eswfall         =  5.4450E-13
  etot_qrise      =  2.5573E-12  from=  1.6000E-08     to=  1.9000E-08
  ei_qrise        =  1.3874E-12
  etot_qfall      =  2.4449E-12  from=  4.8000E-08     to=  5.1000E-08
  ei_qfall        =  1.2153E-12
.
.
.
```

5. 3. 4. Dynamic Power Measurements for Tristate Cells

5. 3. 4. 1. Basic Method

As previously mentioned in the previous chapter on Timing and Capacitance Characterization, one aspect of tristate cells which distinguishes them from non-tristate cells is that tristate cells cannot operate properly in isolation, and therefore their output pins must be connected to the output pins of at least one other tristate cell. Therefore, it is not always possible to measure power dissipation of an individual tristate cell.

In general, it is proposed that ***characterizing tristate cells be performed in the same manner characterizing sequential cells*** in that input energies must be measured independently of output energies, and that for measurement of input energies, the transition delays of input waveforms should be well defined.

For example, the circuit setup in characterizing a noninverting tristate buffer may be as in Figure 5.10. The waveform used was as in Figure 5.11.

The waveform is redundant in that there are many transitions not used in power characterization. This occurs as, since the tristate cell characterization was the last characterization performed (as the characterization method was created last), timing and power characterization activities were performed with the same test bench, and hence the waveforms used also contain transitions used only for timing characterization.

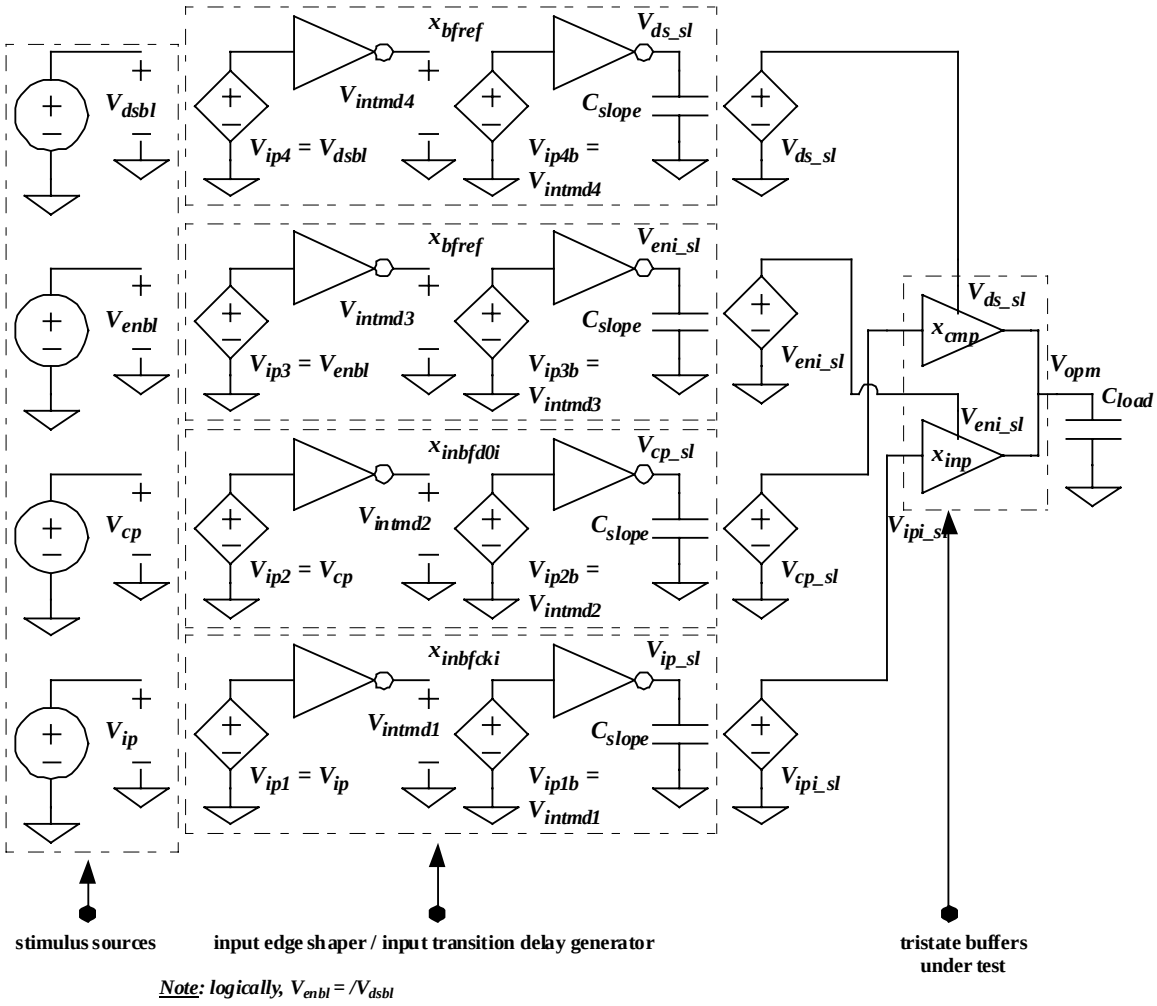


Figure 5. 10. Circuit for Power Characterization of Tristate Buffers

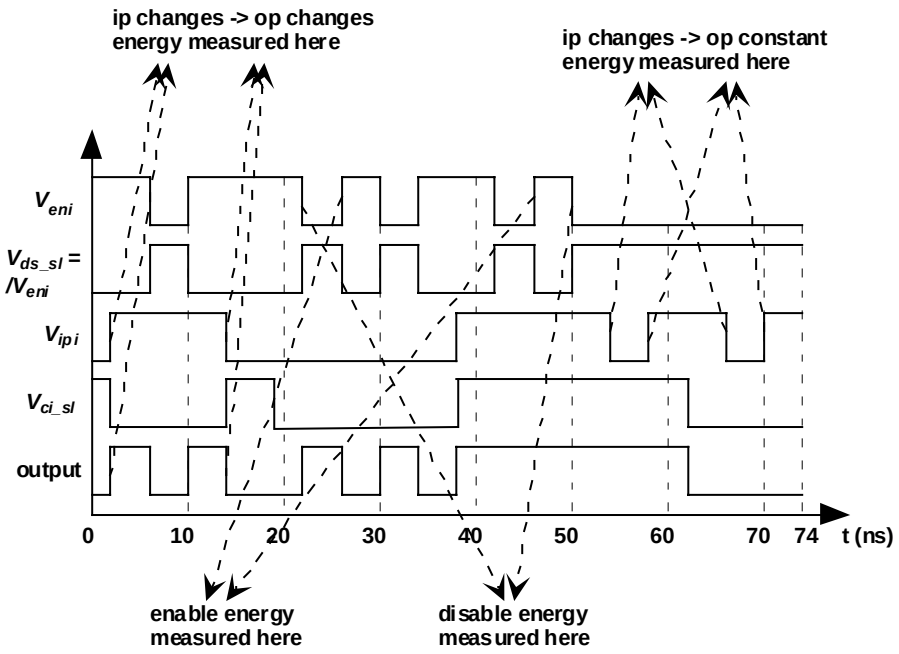


Figure 5. 11. Waveforms for Tristate Buffer Power Characterization

The energy measurement process is as follows:

1. First, input changes → output constant energy (let us call this *ip* energy) is measured.
   i. *ip* rise energy when *op*=low: measured from 70 to 74 ns.
   ii. *ip* rise energy when *op*=high: measured from 58 to 62 ns.
   iii. *ip* rise energy average: taken as average of the above two measurement.
   iv. *ip* fall energy obtained from similar measurement (from 54 ns to 58 ns and from 66 ns to 70 ns, respectively).
2. After *ip* energy is known, energy associated with output changes caused by *ip* changes is calculated via the measurement of *ip* changes → *op* changes energies. This is the only *op* energy measured (i.e. op energy changes due to enable/disable process is assumed to be zero, apart from switching of output capacitances).
   i. *op* rise energy: first, measure energy dissipated from 2 to 6 ns (caused both by *ip* rise event and op rise event), then subtract i*p* rise energy average to obtain energy caused by *op* rise only.

ii. *op* fall energy: first, measure energy dissipated from 14 to 18 ns (caused both by *ip* fall event and op fall event), then subtract *ip* fall energy average to obtain energy caused by *op* fall only.

3. Enable/disable energies could be measured before or after the above two measurement (through en changes → op constant process):
   i. First, enable (*en* rise) → output constant energy (let us call this enable energy) is measured.
   ii. *en* rise energy when *op*=low: measured from 26 to 30 ns.
   iii. *en* rise energy when *op*=high: measured from 46 to 50 ns.
   iv. enable energy average: taken as average of the above two measurement.
   v. *en* fall energy (= disable energy) obtained from similar measurement (from 22 ns to 26 ns and from 42 ns to 46 ns, respectively).

A SPICE file which performs the above measurements may employ the following stimulus specification:

```
** stimuli
vip ip gnd pwl (2ns ss 2.2ns spl 14ns spl 14.1ns ss 38ns ss 38.2ns spl 54ns
spl
+ 54.1ns ss 58ns ss 58.2ns spl 66ns spl 66.1ns ss 70ns ss 70.2ns spl)
** ci for power characterization
vcp cp gnd pwl (2ns spl 2.1ns ss 14ns ss 14.2ns spl 18ns spl 18.1ns ss 38ns
ss
+ 38.2ns spl 62ns spl 62.1ns ss)
.param enbl='spl'
.param dsbl='ss'

** enabler/disabler
venbl en gnd pwl (6ns enbl 6.1ns dsbl 10ns dsbl 10.2ns enbl 22ns enbl 22.1ns
+ dsbl 26ns dsbl 26.2ns enbl 30ns enbl 30.1ns dsbl 34ns dsbl 34.2ns enbl
+ 42ns enbl 42.1ns dsbl 46ns dsbl 46.2ns enbl 50ns enbl 50.1ns dsbl)
vdsbl ds gnd pwl (6ns dsbl 6.2ns enbl 10ns enbl 10.1ns dsbl 22ns dsbl
+ 22.2ns enbl 26ns enbl 26.1ns dsbl 30ns dsbl 30.2ns enbl 34ns enbl 34.1ns
dsbl
+ 42ns dsbl 42.2ns enbl 46ns enbl 46.1ns dsbl 50ns dsbl 50.2ns enbl)
```

Further, given the following subcircuit definitions for edge-degrading buffers:

```
** subcircuit definitions
.subckt nlhiv dd ip op
eip ipi gnd ip gnd 1
```

```
xiv dd ipi op hiv
.ends nlhiv
.subckt nlbuf dd ip op
x1 dd ip ipi nlhiv
x2 dd ipi op nlhiv
.ends nlbuf
.subckt nlhbfzp dd ip en op
eip ipi gnd ip gnd 1
een eni gnd en gnd 1
xbz dd ipi eni op hbfzp
.ends nlhbfzp
```

the edge-degrading buffers and the power measurement circuit may be instantiated as follows:

```
** edge-degraded version of input stimuli
** for finding intrinsic delays of the cells driving the tristate buffers **
xip dd ip ipi nlbuf
xen dd en eni nlbuf
xipn dd ci cii nlbuf
xenn dd ds dsi nlbuf
** for slope delays and slope-dependent power dissipation
xips dd ip ipi_sl nlbuf
cips ipi_sl gnd sloperc
xens dd en eni_sl nlbuf
cens eni_sl gnd sloperc
xcis dd ci ci_sl nlbuf
ccis ci_sl gnd sloperc
xcps dd cp cp_sl nlbuf
ccps cp_sl gnd sloperc
xdss dd dsi ds_sl nlbuf
cdss ds_sl gnd sloperc

*** POWER MEASUREMENTS ***
** Essentially a copy of 1-to-1 enable delay measurement setup **
xinp dd ipi_sl eni_sl opm nlhbfzp
xcmp dd cp_sl ds_sl opm nlhbfzp
cpload opm gnd powerc
```

And since tristate buffers are modeled as having a separate output capacitance (see previous chapter on Timing and Capacitance Characterization), the load driven by the tristate buffer includes also the combined output capacitance of the two buffers. They could be measured as follows:

```
** Cout determination **
vopcd opcd gnd pwl (0ns ss 4ns ss 4.1ns spl 8ns spl 8.2ns ss)
xopcdet1 dd dd gnd opcd hbfzp
xopcdet2 dd gnd gnd opcd hbfzp
.measure qopr integral i(vopcd) from=4ns to=8ns
.measure qopf integral i(vopcd) from=8ns to=12ns
```

```
.measure qop param='(abs(qopr)+abs(qopf))/4'
.measure cout param='qop/spl'

.measure tot_cout param='powerc+(2*cout)' ** actual total load
```

then the measurements are performed by the following statements:

FOR LOAD SWITCHING ENERGIES:

```
** switching energy at output - if at all occuring
.measure esout param='0.5*spl*spl*cout'   *** Eswitch of Cout
.measure elout param='0.5*spl*spl*powerc' *** Eswitch of Cload
.measure esw param='esout+elout'
.measure eswr param='esw'
.measure eswf param='esw'
```

FOR ip ENERGIES:

```
** input energy - no output changes (disabled)
.measure eipr_l integral p(xinp.xbz) from=70ns to=74ns
.measure eipf_l integral p(xinp.xbz) from=66ns to=70ns
.measure eipr_h integral p(xinp.xbz) from=58ns to=62ns
.measure eipf_h integral p(xinp.xbz) from=54ns to=58ns
.measure eipr_av param='0.5*(eipr_l+eipr_h)'
.measure eipf_av param='0.5*(eipf_l+eipf_h)'
.measure eipr param='eipr_av'  **** FINAL RESULTS - as synopsys expects
.measure eipf param='eipf_av'
```

FOR op ENERGIES:

```
** ip -> op energy
** op rise
.measure eiiport integral p(xinp.xbz) from=2ns to=6ns
.measure eiipor param='eiiport-elout-(2*esout)-eipr' ** FINAL RESULT
** op fall
.measure eiipoft integral p(xinp.xbz) from=14ns to=18ns
.measure eiipof param='eiipoft-elout-(2*esout)-eipf' ** FINAL RESUL
```

FOR enable / disable ENERGIES:

```
** enable/disable energy - output low
** enable
.measure eenl integral p(xinp.xbz) from=26ns to=30ns
** disable
.measure edsl integral p(xinp.xbz) from=30ns to=34ns
** enable/disable energy - output high
** enable
.measure eenh integral p(xinp.xbz) from=46ns to=50ns
** disable
.measure edsh integral p(xinp.xbz) from=50ns to=54ns
** enable / disable energy - average (FINAL RESULTS)
.measure eenav param='0.5*(eenh+eenl)'
```

```
.measure edsav param='0.5*(edsh+edsl)'
```

The output file contains the following lines, among others (comments added here).

```
  eipr_l=  4.8131E-13  from=  7.0000E-08     to=  7.4000E-08
** ip rise energy, output low
  eipf_l=  4.8619E-13  from=  6.6000E-08     to=  7.0000E-08
** ip fall energy, output low
  eipr_h=  4.8142E-13  from=  5.8000E-08     to=  6.2000E-08
** ip rise energy, output high
  eipf_h=  5.4265E-13  from=  5.4000E-08     to=  5.8000E-08
** ip fall energy, output high
  eipr_av=  4.8137E-13    ** average rise energy
  eipf_av=  5.1442E-13    ** average fall energy
.
.
.
  eiipor=  3.8997E-13  ** op rise energy
.
.
  eiipof=  3.8024E-13  ** op fall energy
.
.
.
  eenl=  3.5112E-13  from=  2.6000E-08     to=  3.0000E-08
** enable energy, output low
  edsl=  3.2793E-13  from=  3.0000E-08     to=  3.4000E-08
** disable energy, output low
  eenh=  4.0107E-13  from=  4.6000E-08     to=  5.0000E-08
** enable energy, output high
  edsh=  4.0362E-13  from=  5.0000E-08     to=  5.4000E-08
** disable energy, output high
  eenav=  3.7609E-13    ** average enable energy
  edsav=  3.6578E-13    ** average disable energy
```

To create a SYNOPSYS power model, the measurement process is repeated for several different values of load capacitor and input transition delays. Due to relatively lower accuracy caused by the need to operate at least two devices driving the same node, relatively cruder granularity of load values are used here than for non-tristate gates. The conditions used are:

For 0.5µm cells:

input edge degrader use hiv

For ip, en energies:

Input transition delay generated with 0fF, 35fF, 75fF, 150fF, 500fF, 1000fF load.

Tristate buffer drives 75fF load capacitor.

For output energies:

Input transition delay generated with 75fF load.

Tristate buffers drive 0fF, 100fF, 250fF, 500fF, and 1000fF load capacitors.

For 0.35µm cells:

input edge degrader use winv_1

For ip, en energies:

Input transition delay generated with 0fF, 35fF, 75fF, 150fF, 500fF, 1000fF load.

Tristate buffer drives 50fF load capacitor.

For output energies:

Input transition delay generated with 50fF load.

Tristate buffers drive 0fF, 100fF, 250fF, 500fF, and 1000fF load capacitors.

Note that the actual value of load capacitance driven by the buffer is ($C_{load} + 2 \times C_{out}$), where C_{out} is the output capacitance of one buffer.

5. 3. 4. 2. SPICE Example – Complete File

As an example, the following file was used for characterizing the `hbfzp` 0.5µm tristate buffer.

```
hbfzp Power Characterization Measurements

.lib "hp05um_model" cmos_models
.inc hiv.sp
.inc hbfzp.l

** Power Supply
.param spl='3.3v'
.param ss='0v'
.param loaderc='75fF'
.param sloperc='75fF'
.param powerc='100fF'
vdd dd gnd spl

** subcircuit definitions
.subckt nlhiv dd ip op
eip ipi gnd ip gnd 1
xiv dd ipi op hiv
```

```
.ends nlhiv
.subckt nlbuf dd ip op
x1 dd ip ipi nlhiv
x2 dd ipi op nlhiv
.ends nlbuf
.subckt nlhbfzp dd ip en op
eip ipi gnd ip gnd 1
een eni gnd en gnd 1
xbz dd ipi eni op hbfzp
.ends nlhbfzp

** stimuli
vip ip gnd pwl (2ns ss 2.2ns spl 14ns spl 14.1ns ss 38ns ss 38.2ns spl 54ns
spl
+ 54.1ns ss 58ns ss 58.2ns spl 66ns spl 66.1ns ss 70ns ss 70.2ns spl)
** ci for power characterization
vcp cp gnd pwl (2ns spl 2.1ns ss 14ns ss 14.2ns spl 18ns spl 18.1ns ss 38ns
ss
+ 38.2ns spl 62ns spl 62.1ns ss)
.param enbl='spl'
.param dsbl='ss'

** enabler/disabler
venbl en gnd pwl (6ns enbl 6.1ns dsbl 10ns dsbl 10.2ns enbl 22ns enbl 22.1ns
+ dsbl 26ns dsbl 26.2ns enbl 30ns enbl 30.1ns dsbl 34ns dsbl 34.2ns enbl
+ 42ns enbl 42.1ns dsbl 46ns dsbl 46.2ns enbl 50ns enbl 50.1ns dsbl)
vdsbl ds gnd pwl (6ns dsbl 6.2ns enbl 10ns enbl 10.1ns dsbl 22ns dsbl
+ 22.2ns enbl 26ns enbl 26.1ns dsbl 30ns dsbl 30.2ns enbl 34ns enbl 34.1ns
dsbl
+ 42ns dsbl 42.2ns enbl 46ns enbl 46.1ns dsbl 50ns dsbl 50.2ns enbl)
** edge-degraded version of input stimuli
** for finding intrinsic delays of the cells driving the tristate buffers **
xip dd ip ipi nlbuf
xen dd en eni nlbuf
xipn dd ci cii nlbuf
xenn dd ds dsi nlbuf
** for slope delays and slope-dependent power dissipation
xips dd ip ipi_sl nlbuf
cips ipi_sl gnd sloperc
xens dd en eni_sl nlbuf
cens eni_sl gnd sloperc
xcis dd ci ci_sl nlbuf
ccis ci_sl gnd sloperc
xcps dd cp cp_sl nlbuf
ccps cp_sl gnd sloperc
xdss dd dsi ds_sl nlbuf
cdss ds_sl gnd sloperc

** Cout determination **
vopcd opcd gnd pwl (0ns ss 4ns ss 4.1ns spl 8ns spl 8.2ns ss)
xopcdet1 dd dd gnd opcd hbfzp
xopcdet2 dd gnd gnd opcd hbfzp
.measure qopr integral i(vopcd) from=4ns to=8ns
.measure qopf integral i(vopcd) from=8ns to=12ns
.measure qop param='(abs(qopr)+abs(qopf))/4'
.measure cout param='qop/spl'
```

```
** measurement of transition delays of drivers **
.measure tr_bi trig v(xip.ipi) val='0.5*spl' fall=1 targ v(ipi) val='0.5*spl'
+ rise=1
.measure tf_bi trig v(xip.ipi) val='0.5*spl' rise=1 targ v(ipi) val='0.5*spl'
+ fall=1
.measure tr_bs trig v(xips.ipi) val='0.5*spl' fall=1 targ v(ipi_sl)
+ val='0.5*spl' rise=1
.measure tf_bs trig v(xips.ipi) val='0.5*spl' rise=1 targ v(ipi_sl)
+ val='0.5*spl' fall=1
.measure dtr param='tr_bs-tr_bi'
.measure dtf param='tf_bs-tf_bi'

*** POWER MEASUREMENTS ***
** Essentially a copy of 1-to-1 enable delay measurement setup **
xinp dd ipi_sl eni_sl opm nlhbfzp
xcmp dd cp_sl ds_sl opm nlhbfzp
cpload opm gnd powerc

.measure tot_cout param='powerc+(2*cout)' ** actual total load

** switching energy at output - if at all occuring
.measure esout param='0.5*spl*spl*cout'   *** Eswitch of Cout
.measure elout param='0.5*spl*spl*powerc' *** Eswitch of Cload
.measure esw param='esout+elout'
.measure eswr param='esw'
.measure eswf param='esw'

** enable/disable energy - output low
** enable
.measure eenl integral p(xinp.xbz) from=26ns to=30ns
** disable
.measure edsl integral p(xinp.xbz) from=30ns to=34ns
** enable/disable energy - output high
** enable
.measure eenh integral p(xinp.xbz) from=46ns to=50ns
** disable
.measure edsh integral p(xinp.xbz) from=50ns to=54ns
** enable / disable energy - average (FINAL RESULTS)
.measure eenav param='0.5*(eenh+eenl)'
.measure edsav param='0.5*(edsh+edsl)'

** input energy - no output changes (disabled)
.measure eipr_l integral p(xinp.xbz) from=70ns to=74ns
.measure eipf_l integral p(xinp.xbz) from=66ns to=70ns
.measure eipr_h integral p(xinp.xbz) from=58ns to=62ns
.measure eipf_h integral p(xinp.xbz) from=54ns to=58ns
.measure eipr_av param='0.5*(eipr_l+eipr_h)'
.measure eipf_av param='0.5*(eipf_l+eipf_h)'
.measure eipr param='eipr_av'  **** FINAL RESULTS - as synopsys expects
.measure eipf param='eipf_av'

** ip -> op energy
** op rise
.measure eiiport integral p(xinp.xbz) from=2ns to=6ns
.measure eiipor param='eiiport-elout-(2*esout)-eipr' ** FINAL RESULT
** op fall
.measure eiipoft integral p(xinp.xbz) from=14ns to=18ns
```

```
.measure eiipof param='eiipoft-elout-(2*esout)-eipf' ** FINAL RESULT

.option post nomod accurate
.option dcstep=1e-3 gmindc=1e-12
.tran 0.025ns 74ns

.alter load=0fF ** in addition to Cout of tristate buffers themselves
.param powerc='0ff'
.param sloperc='75fF'
.alter load=100fF
.param powerc='100fF'
.param sloperc='75fF'
.alter load=250fF
.param powerc='250fF'
.param sloperc='75fF'
.alter load=500fF
.param powerc='500fF'
.param sloperc='75fF'
.alter load=1000fF
.param powerc='1000fF'
.param sloperc='75fF'

.alter sloper=0fF
.param powerc='75fF'
.param sloperc='0fF'
.alter sloper=35fF
.param powerc='75fF'
.param sloperc='35fF'
.alter sloper=75fF
.param powerc='75fF'
.param sloperc='75fF'
.alter sloper=150fF
.param powerc='75fF'
.param sloperc='150fF'
.alter sloper=500fF
.param powerc='75fF'
.param sloperc='500fF'
.alter sloper=1000fF
.param powerc='75fF'
.param sloperc='1000fF'

.end
```

The output file include the following lines :

```
.
.
.
  qopr= -2.0359E-13  from=  4.0000E-09     to=  8.0000E-09
  qopf=  1.9649E-13  from=  8.0000E-09     to=  1.2000E-08
  qop=  1.0002E-13
```

```
  cout=  3.0309E-14
  tr_bi=  5.7215E-11  targ=  2.2023E-09   trig=  2.1451E-09
  tf_bi=  4.2298E-11  targ=  1.4154E-08   trig=  1.4112E-08
  tr_bs=  1.3488E-10  targ=  2.2799E-09   trig=  2.1451E-09
  tf_bs=  8.9620E-11  targ=  1.4202E-08   trig=  1.4112E-08
  dtr=  7.7663E-11
  dtf=  4.7322E-11
  tot_cout=  6.0618E-14
  esout=  1.6503E-13
  elout=  0.0000E+00
  esw=  1.6503E-13
  eswr=  1.6503E-13
  eswf=  1.6503E-13
  eenl=  3.5112E-13  from=  2.6000E-08     to=  3.0000E-08
  edsl=  3.2793E-13  from=  3.0000E-08     to=  3.4000E-08
  eenh=  4.0107E-13  from=  4.6000E-08     to=  5.0000E-08
  edsh=  4.0362E-13  from=  5.0000E-08     to=  5.4000E-08
  eenav=  3.7609E-13
  edsav=  3.6578E-13
  eipr_l=  4.8131E-13  from=  7.0000E-08     to=  7.4000E-08
  eipf_l=  4.8619E-13  from=  6.6000E-08     to=  7.0000E-08
  eipr_h=  4.8142E-13  from=  5.8000E-08     to=  6.2000E-08
  eipf_h=  5.4265E-13  from=  5.4000E-08     to=  5.8000E-08
  eipr_av=  4.8137E-13
  eipf_av=  5.1442E-13
  eipr=  4.8137E-13
  eipf=  5.1442E-13
  eiiport=  1.2014E-12  from=  2.0000E-09     to=  6.0000E-09
  eiipor=  3.8997E-13
  eiipoft=  1.2247E-12  from=  1.4000E-08     to=  1.8000E-08
  eiipof=  3.8024E-13
.
.
.
```

Chapter VI

Example Design – Twin 4-bit Counter

As an example of the use of the strandard cell library constructed in accordance to our guidelines, the following VHDL code had been synthesized. This code describes a twin 4-bit counter.

```
Library IEEE;
use IEEE.std_logic_1164.all;
use work.all;

entity TOP_COUNT is
  port(CLK,CON,RESET: in std_logic;
       COUNT1: buffer std_logic_vector(3 downto 0);
       COUNT2: buffer std_logic_vector(3 downto 0));
end;

architecture STRUCT of TOP_COUNT is
      component SMB_COUNT
            port(CLK,CON,RESET: in std_logic;
                  COUNT: buffer std_logic_vector(3 downto 0));
      end component;
begin
      U0 :SMB_COUNT Port Map (CLK, CON, RESET, COUNT1);
      U1 :SMB_COUNT Port Map (CLK, CON, RESET, COUNT2);

end STRUCT;

library IEEE;
use IEEE.std_logic_1164.all;
package FINC is

function INC(X :STD_logic_VECTOR) return std_logic_VECTOR;

end FINC;
package body FINC is
  function INC(X : std_logic_VECTOR) return std_logic_VECTOR is
   variable XV: std_logic_VECTOR(X'LENGTH-1 downto 0);
  begin
   XV := X;
   for I in 0 to XV'HIGH loop
    if XV(I) = '0' then
      XV(I) := '1';
      exit;
    else XV(I) := '0';
    end if;
   end loop;
   return XV;
  end INC;
end FINC;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.finc.all;
entity SMB_COUNT is
  port(CLK,CON,RESET: in std_logic;
       COUNT: buffer std_logic_vector(3 downto 0));
end SMB_COUNT;

architecture ALG of SMB_COUNT is
  begin
  process(CLK,CON,RESET)
    begin
      if RESET = '1' then
      COUNT <= "0000";
      elsif CLK'EVENT and CLK='1' then
      if CON = '1' then
                  COUNT <= INC(COUNT);
      end if;
      end if;
  end process;
end ALG;
```

The resulting circuit is shown in the layouts in the following page. The entire design fits into a 1mm×1mm dice. The large dice is necessitated by the use of 13 I/O pads (8 for the "COUNT" outputs, 3 for inputs, and 2 for Power and ground), and those large pads (300μm×90μm) actually occupy most of the die. The core cells actually occupy only approximately only 16% of the core area; it should realistically be possible to reduce core area fourfold or fivefold if the design had, instead of being standalone, formed a part of a larger design (and hence the pads may not be needed).

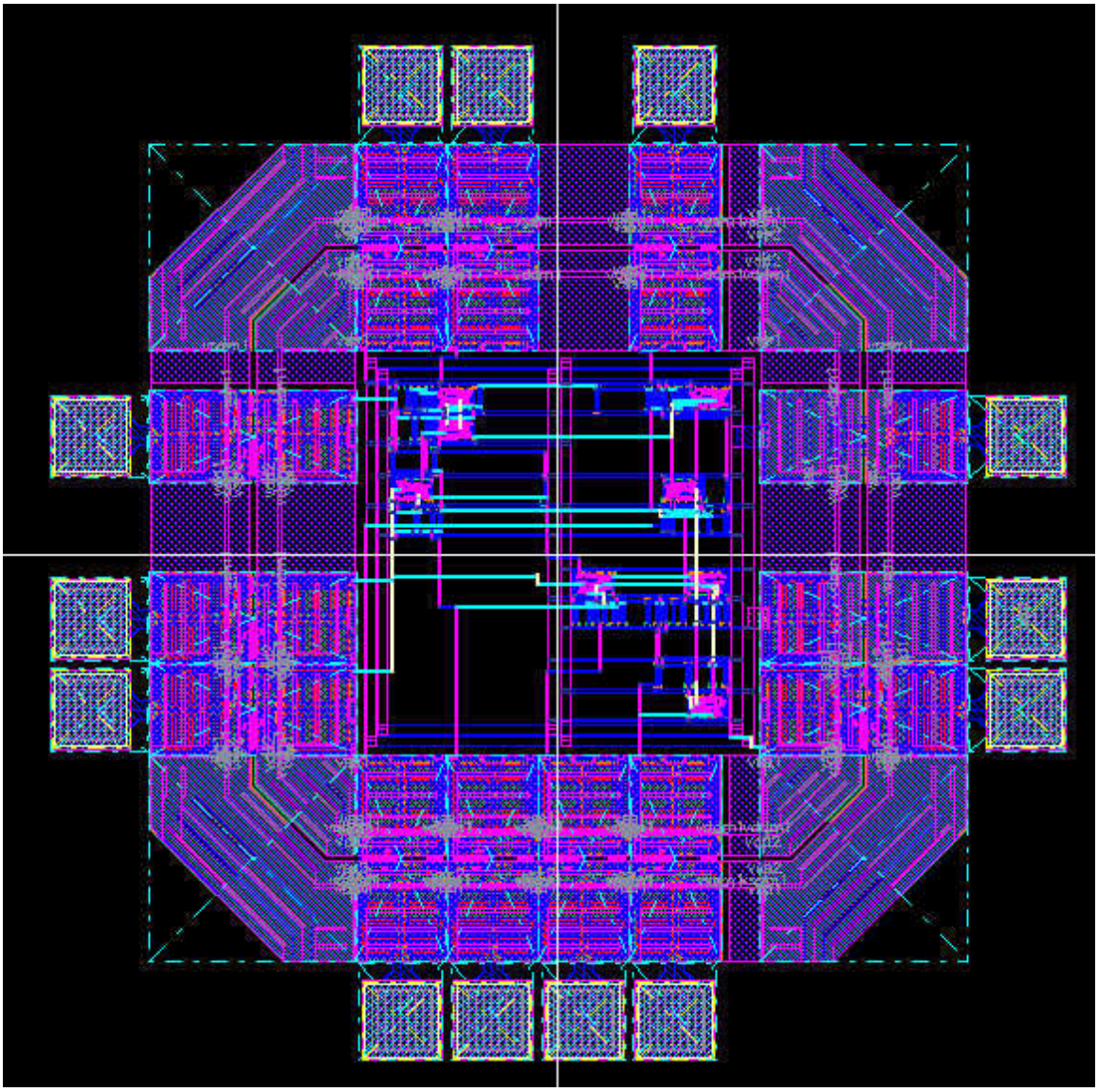


Figure 6. 1. Layout of the circuit produced.

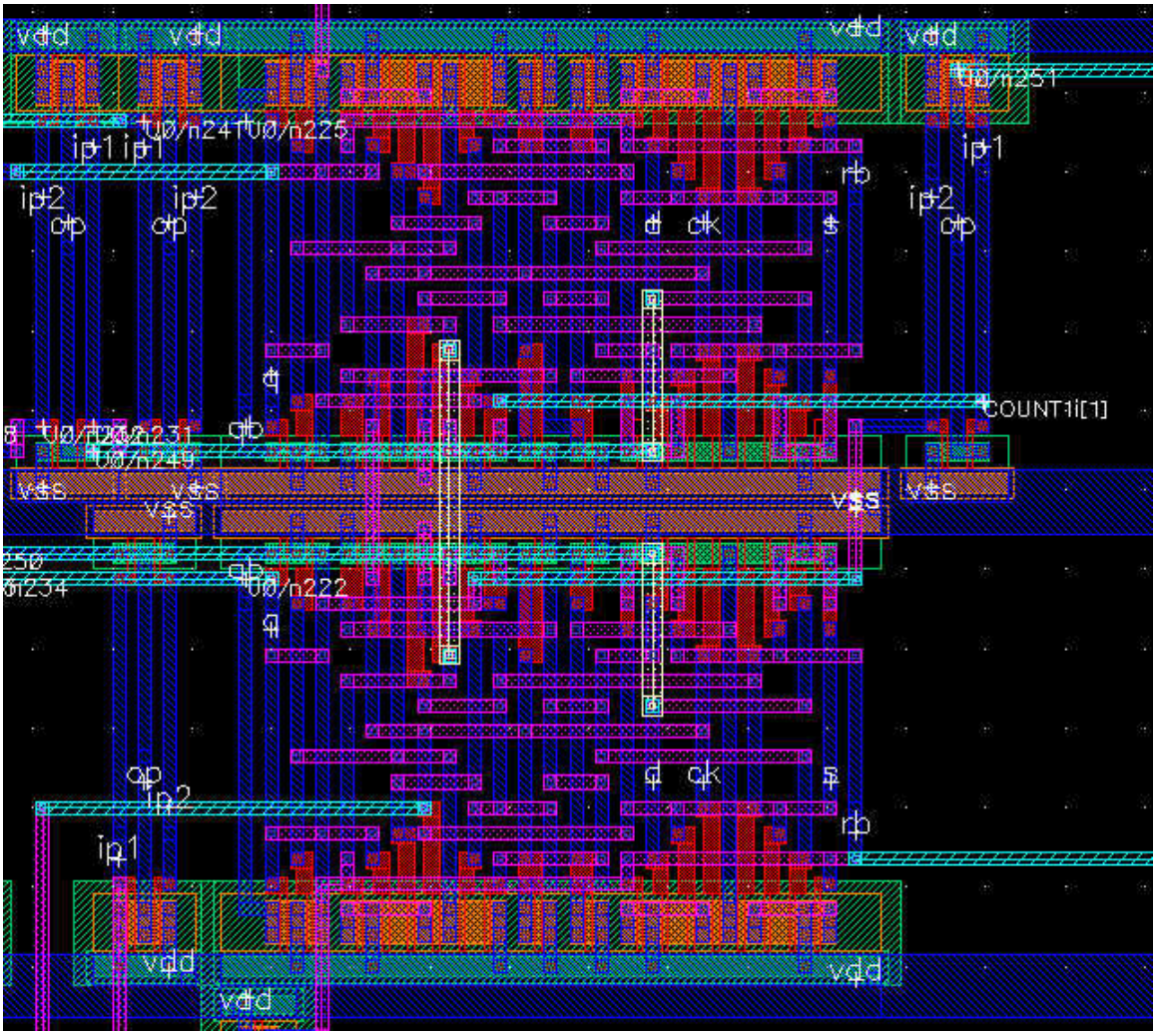


Figure 6. 2. Part of the layout, showing two adjacent rows with shared ground (vss) rail. The large cells in the right are D flip-flops with set and reset. Also shown are 2-input NAND gates.

Chapter VII

Summary

In this work, a set of guidelines for creating layout library for use with Synopsys synthesis and simulation tools and Cadence Placement-and-Routing tools has been presented. Further, a test circuit, namely a twin 4-bit counter, has been built using the resulting cell, hence demonstrating the validity of the guidelines. Further improvements are still possible, however, as the resulting cells may still be minimized in size, or the clock wire width may be increased to allow for longer clock tracks, and hence larger designs. This matters not for our small example design, whose minimal size is dictated by the number of I/O pads; however, such improvements may well be helpful for designs of more realistic, much larger scale as normally encountered in practice.

Additionally, a set of procedure for characterizing CMOS logic cells, for both their timing and power dissipation parameters, are also presented. Since one goal of the work was to devise a set of procedures which are both relatively simple to perform and relatively accurate, the linear timing model is adopted, although this will limit the usability of the resulting procedure to CMOS technology, which should not greatly reduce its value as the CMOS technology is presently the most commonly used technology. Further improvements are still possible, however. One possible area of improvement is in automatizing the entire characterization process. Another area is in using a fully lookup-table timing model instead of the linear model, which would enable the use of the characterization method on other technologies. The larger number of simulations needed to properly characterize a cell using the lookup table model, however, may or may not justify such an attempt.

Bibliography

1. Cadence Corp., *Envisia™ LEF/DEF Language Reference*, 1999
2. M. Cheng, M. Irwin, K. Li, and W. Ye, "Power Characterization of Functional Units," *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers, IEEE*, Vol.1, pp. 775-779, 1999
3. M. A. Cirit, "Characterizing a VLSI Standard Cell Library," *Proceedings of Custom Integrated Circuit Conference, IEEE*, pp. 25.7.1-25.7.4, 1991
4. J.F. Croix and D.F. Wong, "A Fast and Accurate Technique to Optimize Characterization Tables for Logic Synthesis," *Proceedings of Design Automation Conference, IEEE*, pp. 337-340, 1997
5. K. Eshraghian and N.H.E. Weste, "*Principles of CMOS VLSI Design: A System Perspective*," Addison-Wesley Publishing Company, 1994
6. Jing-Yang Jou, Jing-Yuan Lin and Wen-Zen Shen, "*A Structure-Oriented Power Modeling Technique for Macrocells*," IEEE Transactions on VLSI, Vol. 7 No. 3, pp. 380-391, 1999
7. Jing-Yang Jou, Jing-Yuan Lin and Wen-Zen Shen, "*A Power Modeling and Characterization Method for the CMOS Standard Cell Library*," Digest of Technical Papers, International Conference on Computer Aided Design, IEEE, pp. 400-404, 1990
8. Dhimant Patel, "CHARMS: Characterization and Modeling System for Accurate Delay Prediction of ASIC Designs," *Proceedings of Custom Integrated Circuit Conference, IEEE*, pp. 9.5.1-9.5.6, 1990
9. Synopsys Inc, *Library Compiler User Guide, Volume II*, Chapters I – III, 1999

Appendix I

Instructions for LEF File Generation Process

The abstract generator program *abstract* is used for generating LEF files. However, it is used only for generating the part of LEF files which partially describes the geometry of the cells, and not the complete LEF file. The technology description part of the LEF file must be created manually.

A LEF file describing a library has two parts:

1. The technology description part, describing:
   i. The layers available in the technology. Only layers involved in the PNR process need to be included.
   ii. Part of design rules which affects PNR operation, such as minimum metal width and separation. It does not usually have to include, for example, rules for separation between n-wells, as wells are not used as wires and usually cells are designed such that no well separation violation should occur even between adjacent cells.
   iii. Library designer-defined routing rules, such as the chosen value of routing pitch, the preferred directions for metal tracks, or the geometric description of the via used (here the term “via” is used to also include the metal extensions needed in both layers of metals connected to the cut layer).
   iv. (optional) Electrical properties of the layers in the library, such as maximum current per cut, unit square resistance for the metal layer used.

2. The cell description part, describing the geometries comprising each cell:
   i. The shape and size of cells, as defined by their respective boundaries
   ii. The location of pins, and the layer those pins rest on, as well as geometric description of other shapes in the same nodes

iii. Detailed descriptions of obstructions, namely shapes in conducting layers which do not belong to any particular pins, but prohibit the passage of routing tracks in the same layer.

The first part (technology description) must be generated either manually or with some other tool. It is required by *abstract* as an input for generation of the second part. The generation of geometric description of even a cell of modest complexity, such as a flip-flop with set/reset, is tedious and error-prone if performed manually, hence the use of *abstract* or other similar tools is mandatory.

Steps involved in generating LEF file:

1. Creation of technology description part of LEF file
2. Converting the LEF file into Envisia technology file using *abstract*.
3. Importing layout (in GDS format) and logical description of cell (in Verilog) into *abstract*.
4. Generating the complete LEF file.

In detail, this is performed with the following steps, which could also found in **<cadence-installation-dir>/<SE-installation-dir>/doc/abstract/abstractTOC.tgf**, which could be viewed using Cadence help browser *openbook*. The first two steps (for creating DPUX file from LEF file) are unnecessary for those skilled in creating dpux file; however, the syntax of DPUX is more complicated than that of LEF.

Have the LEF file ready. Refer to *openbook* document for information on LEF syntax, *Envisia™ LEF/DEF Language Reference* at **<cadence-installation-dir>/<SE-installation-dir>/doc/ASICpnr/lefdefref/lefdefrefTOC.obk**. A sample LEF file will also be included at the end of this document.

Envisia technology file generation

It is assumed that no files with extension `.dpux` is present. By default, an abstract technology file is named `tech.dpux` – this name is not arbitrary.

1. open the abstract generator with the following command in the UNIX prompt:
   abstract &
2. The program *abstract* appears not to have been a fully finished product. Here, it will display a rather cryptic message window, saying:
   "There are errors in technology data. Please fix these before continuing with abstract."
   Press OK. This message here arrives because *abstract* is not instructed to read a `tech.dpux` file, or abstract technology file. Which is just fine, for we intend to create one instead.
3. Choose **File → Technology**
   A new window (Technology File Editor) appears. Choose **File → Import LEF file**, and choose the technology LEF file already prepared.
4. Choose **Category : Library Path**
5. Press **Add** button – then specify a path in which your abstract library will be stored. If the directory specified does not exist *abstract* will create one. **NOTE:** This library is NOT the same as the dfII/icfb layout library, and must not be stored in the same directory to avoid the actual layout from getting overwritten with abstract generator data which is not readable by the layout drawing tool and totally unusable for physical layout.
6. Choose **Categories : Layer**
7. Choose **Mapping** tab
   Empty the map by selecting the number associated with the entry and press **Delete** button
   Then press **Map** button and import the GDS2 mapping file
8. Select the **Define** tab and define the classes of poly, contact, via, and metal layers if not already defined in the technology LEF file (apparent as their class will be shown as "Unknown"). For other layers, which normally will not be used for routing, it is not critical that their classes should be defined, and indeed better left UNKNOWN rather than defined incorrectly.
9. Map definition is complete, check the correctness of grid rules, select **Categories : Grid**, check whether the values are as intended. *abstract* seems to tend to get it wrong here. In particular, *offset* and *routing grid* are often incorrect. *Routing grid* is another term for pitch.

10. After it is finished, save the file: **File → Save as** ..... `tech.dpux`. This name is NOT arbitrary.
11. Close Technology Editor window: **File → Close**.
12. Exit by closing the Abstract window (the ornate one – not the plain vanilla one) with **File → Exit**. Do not attempt to exit by closing the PCW / log info window first.

Complete LEF file generation

13. Open *abstract* again, with the following command:

    abstract –tech . &

    This will instruct *abstract* to read the `tech.dpux` file in the working directory – this is why the name `tech.dpux` was chosen, here it is not arbitrary.

    All operations should be done on Abstract window, and not on PCW window.
14. Choose **File → Library**. This should open the abstract library, or create one if it does not already exist.
15. Import the layout file (in GDSII format) : **File → Import → Layout**. Import the GDSII layout file. Choose "No Mapping" – this will preserve the case in the names of the instances inside the cells. Note that this works best if the cells are flattened.
16. Import the logic information (Verilog or TLF) file : **File → Import → Logical**. Note that the format must be specified correctly.
17. Select the cells for which LEF files need to be generated. To select all of them use **Select → All**.

Defining pins:

18. Select **Flow → Pins**. Then in the new window specify the layers in which labels for pins are drawn. For example, if your pins are all metal1 pins and they are given label in metal2, specify (metal2 metal1) in the topmost text box. The statement means that for all labels in metal2, use the name specified by the label for the metal1 shape over which the label's origin is placed.

19. Select **Boundary** tab – and check whether any boundary location adjustments are needed. For example, if the boundary needs to be located 0.4μm above the top and lower than the bottom of the power pins (or the topmost/bottom metal shapes), enter the value 0.4 to the **Top** and **Bottom** input text boxes in the **Adjust Boundary By** input frame. This will stretch the cell boundary (relative to metal boundary) accordingly.
20. Press **Run** button and wait until pin generation step is completed. Most likely, there will be warnings for parts of the cells being outside the boundary (due to parts of n- or p-well which lies outside the border). They could be just ignored, as they will be abutted with the wells of the same polarity inside the adjacent cells during the placement step.

Pin and obstruction extraction:

21. Choose **Flow → Extract → Run**.

Abstract generation:

22. Choose **Flow → Abstract**.
23. Choose the **Overlap** tab.
24. Choose **as needed**.
25. Choose the **Grid** tab – and correct any error in offset or routing grid if necessary. Seems like abstract tends to get it wrong here.
26. Press **Run**.

Verification inside the abstract generation environment:

It is prudent to check whether the library is actually routable. Choose **Flow → Verify**. This command will invoke Silicon Ensemble and construct a test circuit to test the routability of the LEF files constructed from those cells. It will warn of potential problems.

Postprocessing:

For all pins that are used for power or ground, the SHAPE must be defined. The only exception is the supply / ground for core cells provided by power or ground pads.

For core cells:

The SHAPE property should be defined as ABUTMENT. Hence, lines such as:

USE POWER ;

or

USE GROUND ;

should become

USE POWER ; SHAPE ABUTMENT ;

or

USE GROUND ; SHAPE ABUTMENT ;

respectively. This is usually already performed automatically by abstract generator. If not, it has to be performed manually.

For pad or corner cells:

This is almost never performed manually by the abstract generator. Here the correct property shape for VDD/ground ring pin is FEEDTHRU. Hence, the aforementioned two lines become:

USE POWER ; SHAPE FEEDTHRU ;

USE GROUND ; SHAPE FEEDTHRU ;

respectively.

Miscellaneous tips:

- Labels for pins should be in a nonconducting, physically nonexistent layers (such as text) so that would not cause any design rule violations if design rule check (DRC) needs to be performed inside a layout editor session. While such a DRC error is actually meaningless and can simply be ignored, it may lead to confusion.
- Since pad cells are usually very large, if a “pin” layer is not present in the design kit being used, it is likely to be impractical to use a full pad cell for LEF file generation. It is probably better to simply use a dummy cell representing only the outermost edges of pin locations, using thin wires – the same width as specified for normal tracks. The following picture gives an example.

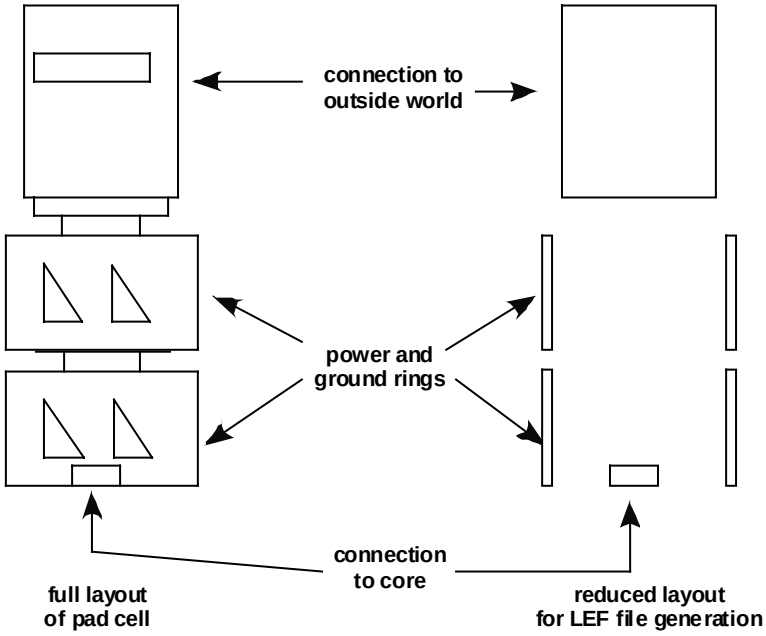


Figure A-1.1. Simplification of pad layout for LEF file generation.

LEF File Examples

Here two LEF FIle Examples are presented. The first example is a *Technology* LEF file. The second one is the part of the macro definition for a D flip-flop produced by abstract (layout to be provided).

Technology LEF File Example

```
VERSION 5.3 ;
NAMESCASESENSITIVE ON ;
BUSBITCHARS "[]" ;

UNITS
  DATABASE MICRONS 1000 ;
END UNITS

LAYER nwell
  TYPE      MASTERSLICE ;
END nwell

LAYER pwell
  TYPE      MASTERSLICE ;
```

```
END pwell

LAYER cwell
  TYPE      MASTERSLICE ;
END cwell

LAYER pbase
  TYPE      MASTERSLICE ;
END pbase

LAYER active
  TYPE      MASTERSLICE ;
END active

LAYER tactive
  TYPE      MASTERSLICE ;
END tactive

LAYER ccd
  TYPE      MASTERSLICE ;
END ccd

LAYER nselect
  TYPE      MASTERSLICE ;
END nselect

LAYER pselect
  TYPE      MASTERSLICE ;
END pselect

LAYER poly
  TYPE      MASTERSLICE ;
END poly

LAYER polycap
  TYPE      MASTERSLICE ;
END polycap

LAYER elec
  TYPE      MASTERSLICE ;
END elec

LAYER glass
  TYPE      MASTERSLICE ;
END glass

LAYER pad
  TYPE      MASTERSLICE ;
END pad

LAYER sblock
  TYPE      MASTERSLICE ;
END sblock

LAYER open
  TYPE      MASTERSLICE ;
END open
```

```
LAYER pstop
  TYPE      MASTERSLICE ;
END pstop

LAYER highres
  TYPE      MASTERSLICE ;
END highres

LAYER m4prime
  TYPE      MASTERSLICE ;
END m4prime

# the following three layers are (ab)used for labeling
# text for core cells
# res_id upper metal of pads (4, 2, others), cap_id metal1 of pads

LAYER text
  TYPE      MASTERSLICE ;
END text

LAYER res_id
  TYPE      MASTERSLICE ;
END res_id

LAYER cap_id
  TYPE      MASTERSLICE ;
END cap_id


LAYER cc
  TYPE      CUT ;
  SPACING   0.60 ;
END cc

LAYER metal1
  TYPE            ROUTING ;
  DIRECTION HORIZONTAL ;
  PITCH           1.6  ;
  WIDTH           0.8 ;
  SPACING   0.8 ;
  RESISTANCE      RPERSQ 0.07 ;
  CAPACITANCE     CPERSQDIST 3.5e-05 ;
END metal1

LAYER via
  TYPE      CUT ;
  SPACING   0.6 LAYER cc ;
END via

LAYER metal2
  TYPE            ROUTING ;
  DIRECTION VERTICAL ;
  PITCH           1.6  ;
  WIDTH           0.8 ;
  SPACING   0.8 ;
  RESISTANCE      RPERSQ 0.07 ;
```

```
  CAPACITANCE     CPERSQDIST 3.5e-05 ;
END metal2

LAYER via2
  TYPE      CUT ;
  SPACING   0.6 ;
END via2

LAYER metal3
  TYPE            ROUTING ;
  DIRECTION HORIZONTAL ;
  PITCH           1.6  ;
  WIDTH           0.8 ;
  SPACING   0.8 ;
  RESISTANCE      RPERSQ 0.07 ;
  CAPACITANCE     CPERSQDIST 3.5e-05 ;
END metal3

LAYER via3
  TYPE      CUT ;
  SPACING   0.8 ;
END via3

LAYER metal4
  TYPE            ROUTING ;
  DIRECTION VERTICAL ;
  PITCH           3.2  ;
  WIDTH           1.2 ;
  SPACING   2.0 ;
  RESISTANCE      RPERSQ 0.04 ;
  CAPACITANCE     CPERSQDIST 4e-05 ;
END metal4

SPACING
  SAMENET cc  cc  0.6 ;
  SAMENET metal1  metal1      0.6 STACK ;
  SAMENET cc  via 0.0 STACK ;
  SAMENET metal2  metal2      0.6 STACK ;
  SAMENET via  via2     0.0 STACK ;
  SAMENET metal3  metal3      0.6 STACK ;
  SAMENET via2  via3    0.0 STACK ;
  SAMENET metal4  metal4      0.6 ;
  SAMENET via  via      0.6 ;
  SAMENET via2  via2    0.6 ;
  SAMENET via3  via3    0.8 ;
END SPACING

VIA M2_M1 DEFAULT
  LAYER metal1 ;
    RECT -0.400 -0.400 0.400 0.400 ;
  LAYER via ;
    RECT -0.200 -0.200 0.200 0.200 ;
  LAYER metal2 ;
    RECT -0.400 -0.400 0.400 0.400 ;
END M2_M1

VIA M3_M2 DEFAULT
```

```
  LAYER metal2 ;
    RECT -0.400 -0.400 0.400 0.400 ;
  LAYER via2 ;
    RECT -0.200 -0.200 0.200 0.200 ;
  LAYER metal3 ;
    RECT -0.400 -0.400 0.400 0.400 ;
END M3_M2

VIA M4_M3 DEFAULT
  LAYER metal3 ;
    RECT -0.400 -0.400 0.400 0.400 ;
  LAYER via3 ;
    RECT -0.200 -0.200 0.200 0.200 ;
  LAYER metal4 ;
    RECT -0.600 -0.600 0.600 0.600 ;
END M4_M3

VIARULE via_array GENERATE
  LAYER metal1 ;
    DIRECTION HORIZONTAL ;
    OVERHANG 0.20 ;
  LAYER metal2 ;
    DIRECTION VERTICAL ;
    OVERHANG 0.20 ;
  LAYER via ;
    RECT -0.20 -0.20 0.20 0.20 ;
    SPACING 1.6 BY 1.6 ;
END via_array

VIARULE via2_array GENERATE
  LAYER metal2 ;
    DIRECTION VERTICAL ;
    OVERHANG 0.4 ;
  LAYER metal3 ;
    DIRECTION HORIZONTAL ;
    OVERHANG 0.4 ;
  LAYER via2 ;
    RECT -0.20 -0.20 0.20 0.20 ;
    SPACING 3.2 BY 3.2 ;
END via2_array

VIARULE via3_array GENERATE
  LAYER metal3 ;
    DIRECTION HORIZONTAL ;
    OVERHANG 0.4 ;
  LAYER metal4 ;
    DIRECTION VERTICAL ;
    OVERHANG 0.4 ;
  LAYER via3 ;
    RECT -0.20 -0.20 0.20 0.20 ;
    SPACING 2.4 BY 2.4 ;
END via3_array

VIARULE TURN1 GENERATE
  LAYER metal1 ;
    DIRECTION HORIZONTAL ;
  LAYER metal1 ;
```

```
    DIRECTION VERTICAL ;
END TURN1

VIARULE TURN2 GENERATE
  LAYER metal2 ;
    DIRECTION HORIZONTAL ;
  LAYER metal2 ;
    DIRECTION VERTICAL ;
END TURN2

VIARULE TURN3 GENERATE
  LAYER metal3 ;
    DIRECTION HORIZONTAL ;
  LAYER metal3 ;
    DIRECTION VERTICAL ;
END TURN3

VIARULE TURN4 GENERATE
  LAYER metal4 ;
    DIRECTION HORIZONTAL ;
  LAYER metal4 ;
    DIRECTION VERTICAL ;
END TURN4

END LIBRARY
```

Macro Example

The following picture is of the D latch previously shown in Figure 3.7, here repeated as it will be used as an example of LEF File Generation.

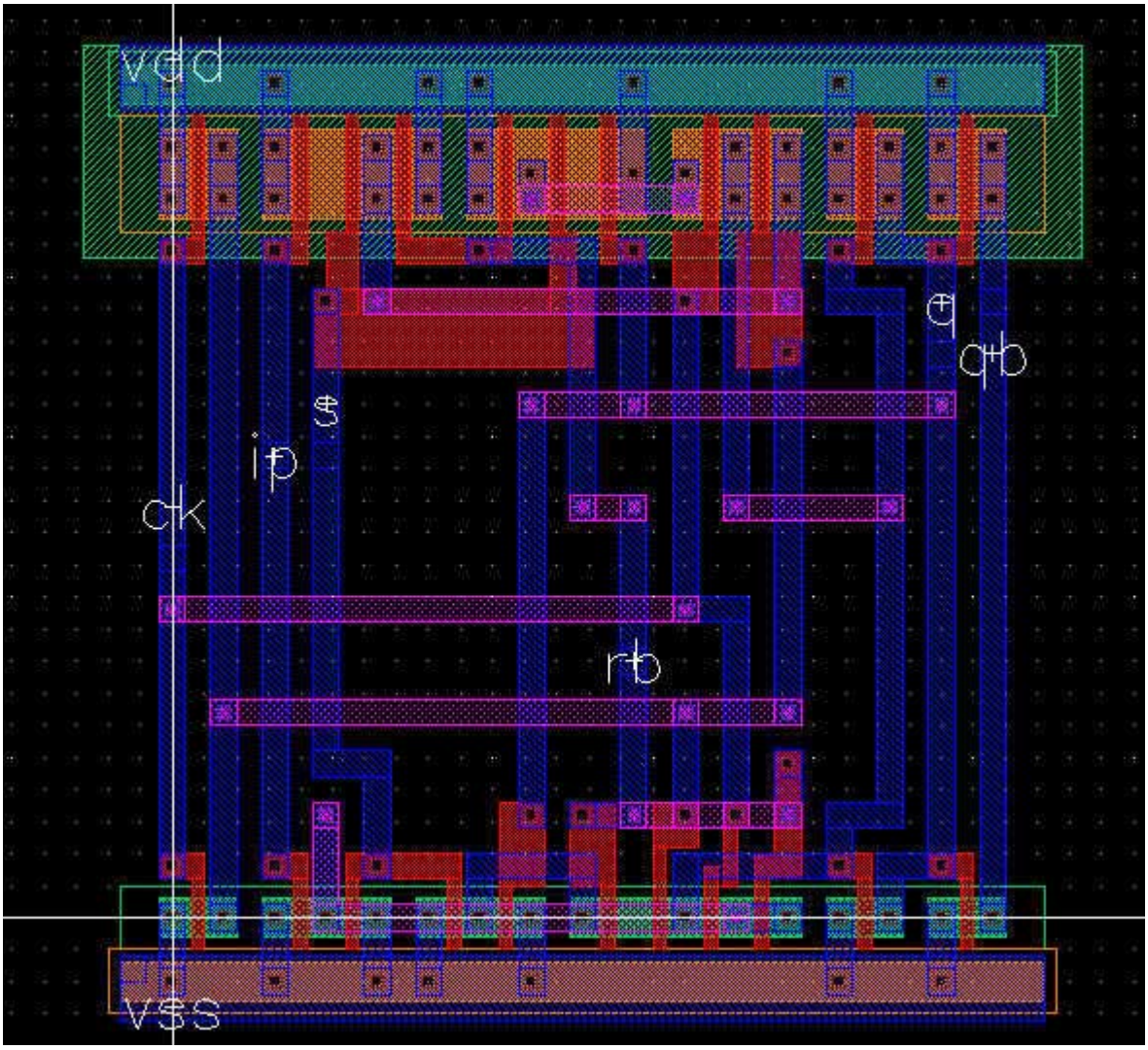


Figure A-1.2. Layout of D Latch

The Resulting LEF macro is as follows:

```
MACRO dlrs
  CLASS  CORE ;
  FOREIGN dlrs -1.600 -4.800 ;
  ORIGIN 1.600 4.800 ;
  SIZE 28.800 BY 33.600 ;
  SYMMETRY X Y  ;
  SITE CoreSite ;
  PIN q
    DIRECTION OUTPUT ;
    PORT
```

```
    LAYER metal2 ;
      RECT 10.800 15.600 24.400 16.400 ;
    LAYER via ;
      RECT 11.000 15.800 11.400 16.200 ;
      RECT 14.200 15.800 14.600 16.200 ;
      RECT 23.800 15.800 24.200 16.200 ;
    LAYER metal1 ;
      RECT 10.800 2.800 11.600 16.400 ;
      RECT 14.000 15.600 14.800 21.200 ;
      RECT 22.000 -0.400 22.800 2.000 ;
      RECT 22.000 20.400 22.800 24.400 ;
      RECT 22.000 1.200 24.400 2.000 ;
      RECT 23.600 1.200 24.400 21.200 ;
      RECT 22.000 20.400 24.400 21.200 ;
  END
END q
PIN s
  DIRECTION INPUT ;
  PORT
    LAYER metal1 ;
      RECT 4.400 4.400 5.200 19.600 ;
      RECT 6.000 1.200 6.800 5.200 ;
      RECT 4.400 4.400 6.800 5.200 ;
  END
END s
PIN qb
  DIRECTION OUTPUT ;
  PORT
    LAYER metal1 ;
      RECT 25.200 -0.400 26.000 24.400 ;
  END
END qb
PIN vss
  DIRECTION INOUT ;
  USE GROUND ;
  SHAPE ABUTMENT ;
  PORT
    LAYER metal1 ;
      RECT -0.400 -3.600 0.400 0.400 ;
      RECT 2.800 -3.600 3.600 0.400 ;
      RECT 6.000 -3.600 6.800 0.400 ;
      RECT 7.600 -3.600 8.400 0.400 ;
      RECT 10.800 -3.600 11.600 0.400 ;
      RECT 20.400 -3.600 21.200 0.400 ;
      RECT 23.600 -3.600 24.400 0.400 ;
      RECT -1.600 -3.600 27.200 -1.200 ;
  END
END vss
PIN ck
  DIRECTION INPUT ;
  PORT
    LAYER metal2 ;
      RECT -0.400 9.200 16.400 10.000 ;
    LAYER via ;
      RECT -0.200 9.400 0.200 9.800 ;
      RECT 15.800 9.400 16.200 9.800 ;
    LAYER metal1 ;
```

```
      RECT -0.400 1.200 0.400 21.200 ;
      RECT 15.600 9.200 16.400 19.600 ;
      RECT 17.200 2.800 18.000 10.000 ;
      RECT 15.600 9.200 18.000 10.000 ;
  END
END ck
PIN rb
  DIRECTION INPUT ;
  PORT
    LAYER metal2 ;
      RECT 12.400 12.400 14.800 13.200 ;
      RECT 14.000 2.800 19.600 3.600 ;
    LAYER via ;
      RECT 12.600 12.600 13.000 13.000 ;
      RECT 14.200 12.600 14.600 13.000 ;
      RECT 14.200 3.000 14.600 3.400 ;
      RECT 19.000 3.000 19.400 3.400 ;
    LAYER metal1 ;
      RECT 12.400 12.400 13.200 21.200 ;
      RECT 9.200 20.400 13.200 21.200 ;
      RECT 12.400 2.800 14.800 3.600 ;
      RECT 14.000 2.800 14.800 13.200 ;
      RECT 18.800 2.800 19.600 5.200 ;
  END
END rb
PIN ip
  DIRECTION INPUT ;
  PORT
    LAYER metal1 ;
      RECT 2.800 1.200 3.600 21.200 ;
  END
END ip
PIN vdd
  DIRECTION INOUT ;
  USE POWER ;
  SHAPE ABUTMENT ;
  PORT
    LAYER metal1 ;
      RECT -0.400 22.000 0.400 27.600 ;
      RECT 2.800 22.000 3.600 27.600 ;
      RECT 7.600 22.000 8.400 27.600 ;
      RECT 9.200 22.000 10.000 27.600 ;
      RECT 14.000 22.000 14.800 27.600 ;
      RECT 20.400 22.000 21.200 27.600 ;
      RECT 23.600 22.000 24.400 27.600 ;
      RECT -1.600 25.200 27.200 27.600 ;
  END
END vdd
OBS
    LAYER cc ;
      RECT 25.400 -0.200 25.800 0.200 ;
      RECT 25.400 22.200 25.800 22.600 ;
      RECT 25.400 23.800 25.800 24.200 ;
      RECT 23.800 -2.200 24.200 -1.800 ;
      RECT 23.800 -0.200 24.200 0.200 ;
      RECT 23.800 1.400 24.200 1.800 ;
      RECT 23.800 20.600 24.200 21.000 ;
```

```
RECT 23.800 22.200 24.200 22.600 ;
RECT 23.800 23.800 24.200 24.200 ;
RECT 23.800 25.800 24.200 26.200 ;
RECT 22.200 -0.200 22.600 0.200 ;
RECT 22.200 22.200 22.600 22.600 ;
RECT 22.200 23.800 22.600 24.200 ;
RECT 20.600 -2.200 21.000 -1.800 ;
RECT 20.600 -0.200 21.000 0.200 ;
RECT 20.600 1.400 21.000 1.800 ;
RECT 20.600 20.600 21.000 21.000 ;
RECT 20.600 22.200 21.000 22.600 ;
RECT 20.600 23.800 21.000 24.200 ;
RECT 20.600 25.800 21.000 26.200 ;
RECT 19.000 -0.200 19.400 0.200 ;
RECT 19.000 4.600 19.400 5.000 ;
RECT 19.000 17.400 19.400 17.800 ;
RECT 19.000 22.200 19.400 22.600 ;
RECT 19.000 23.800 19.400 24.200 ;
RECT 17.400 3.000 17.800 3.400 ;
RECT 17.400 22.200 17.800 22.600 ;
RECT 17.400 23.800 17.800 24.200 ;
RECT 15.800 -0.200 16.200 0.200 ;
RECT 15.800 3.000 16.200 3.400 ;
RECT 15.800 19.000 16.200 19.400 ;
RECT 15.800 23.000 16.200 23.400 ;
RECT 14.200 20.600 14.600 21.000 ;
RECT 14.200 23.000 14.600 23.400 ;
RECT 14.200 25.800 14.600 26.200 ;
RECT 12.600 -0.200 13.000 0.200 ;
RECT 12.600 3.000 13.000 3.400 ;
RECT 11.000 -2.200 11.400 -1.800 ;
RECT 11.000 -0.200 11.400 0.200 ;
RECT 11.000 3.000 11.400 3.400 ;
RECT 11.000 23.000 11.400 23.400 ;
RECT 9.400 -0.200 9.800 0.200 ;
RECT 9.400 20.600 9.800 21.000 ;
RECT 9.400 22.200 9.800 22.600 ;
RECT 9.400 23.800 9.800 24.200 ;
RECT 9.400 25.800 9.800 26.200 ;
RECT 7.800 -2.200 8.200 -1.800 ;
RECT 7.800 -0.200 8.200 0.200 ;
RECT 7.800 22.200 8.200 22.600 ;
RECT 7.800 23.800 8.200 24.200 ;
RECT 7.800 25.800 8.200 26.200 ;
RECT 6.200 -2.200 6.600 -1.800 ;
RECT 6.200 -0.200 6.600 0.200 ;
RECT 6.200 1.400 6.600 1.800 ;
RECT 6.200 22.200 6.600 22.600 ;
RECT 6.200 23.800 6.600 24.200 ;
RECT 4.600 -0.200 5.000 0.200 ;
RECT 4.600 19.000 5.000 19.400 ;
RECT 3.000 -2.200 3.400 -1.800 ;
RECT 3.000 -0.200 3.400 0.200 ;
RECT 3.000 1.400 3.400 1.800 ;
RECT 3.000 20.600 3.400 21.000 ;
RECT 3.000 22.200 3.400 22.600 ;
RECT 3.000 23.800 3.400 24.200 ;
```

```
        RECT 3.000 25.800 3.400 26.200 ;
        RECT 1.400 -0.200 1.800 0.200 ;
        RECT 1.400 22.200 1.800 22.600 ;
        RECT 1.400 23.800 1.800 24.200 ;
        RECT -0.200 -2.200 0.200 -1.800 ;
        RECT -0.200 -0.200 0.200 0.200 ;
        RECT -0.200 1.400 0.200 1.800 ;
        RECT -0.200 20.600 0.200 21.000 ;
        RECT -0.200 22.200 0.200 22.600 ;
        RECT -0.200 23.800 0.200 24.200 ;
        RECT -0.200 25.800 0.200 26.200 ;
      LAYER via ;
        RECT 22.200 12.600 22.600 13.000 ;
        RECT 19.000 6.200 19.400 6.600 ;
        RECT 19.000 19.000 19.400 19.400 ;
        RECT 17.400 -0.200 17.800 0.200 ;
        RECT 17.400 12.600 17.800 13.000 ;
        RECT 15.800 6.200 16.200 6.600 ;
        RECT 15.800 22.200 16.200 22.600 ;
        RECT 11.000 22.200 11.400 22.600 ;
        RECT 6.200 19.000 6.600 19.400 ;
        RECT 4.600 3.000 5.000 3.400 ;
        RECT 1.400 6.200 1.800 6.600 ;
      LAYER metal1 ;
        RECT 20.400 18.800 22.800 19.600 ;
        RECT 22.000 2.800 22.800 19.600 ;
        RECT 20.400 2.800 22.800 3.600 ;
        RECT 20.400 18.800 21.200 21.200 ;
        RECT 20.400 1.200 21.200 3.600 ;
        RECT 15.600 1.200 21.200 2.000 ;
        RECT 15.600 -0.400 16.400 2.000 ;
        RECT 17.200 -0.400 19.600 0.400 ;
        RECT 18.800 6.000 19.600 18.000 ;
        RECT 18.800 18.800 19.600 24.400 ;
        RECT 17.200 12.400 18.000 24.400 ;
        RECT 15.600 2.800 16.400 6.800 ;
        RECT 15.600 22.000 16.400 23.600 ;
        RECT 9.200 1.200 13.200 2.000 ;
        RECT 12.400 -0.400 13.200 2.000 ;
        RECT 9.200 -0.400 10.000 2.000 ;
        RECT 10.800 22.000 11.600 23.600 ;
        RECT 6.000 18.800 6.800 24.400 ;
        RECT 4.400 -0.400 5.200 3.600 ;
        RECT 1.200 -0.400 2.000 24.400 ;
      LAYER metal2 ;
        RECT 17.200 12.400 22.800 13.200 ;
        RECT 1.200 6.000 19.600 6.800 ;
        RECT 6.000 18.800 19.600 19.600 ;
        RECT 4.400 -0.400 18.000 0.400 ;
        RECT 4.400 -0.400 5.200 3.600 ;
        RECT 10.800 22.000 16.400 22.800 ;
  END
END dlrs
```

Vita of Jos Budi Sulistyo

Jos Budi Sulistyo was born in Jember, Indonesia. He obtained his B.S. degree in Electrical Engineering from Texas A&M University in 1993. Following his graduation, he joined the Indonesian National Atomic Energy Agency, where he is currently a staff member.

In August 1998, Jos joined Dr. Ha's research group at the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University. Currently, he plans to continue his education to the PhD level. His current research interest is development of low power standard cell.